University of Alberta

CONNECTION-LESS PARADIGM IN HIGH-SPEED NETWORKING

by

Władysław Olesiński

©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

Department of Computing Science

Edmonton, Alberta
Spring 1999

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-39577-4

Canada

University of Alberta

Library Release Form

Name of Author: Władysław Olesiński

Title of Thesis: Connection-Less Paradigm in High-Speed Networking

Degree: Doctor of Philosophy

Year this Degree Granted: 1999

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

Wladyslaw Olesinski

Wladysław Olesiński
Apt. 7, 9827-104 St
Edmonton, Alberta T5K 0Y8

Date: Nov 26, 1998

Begin at the beginning and go on till you come to the end:
then stop.
Lewis Carroll, "Alice in Wonderland"

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Connection-Less Paradigm in High-Speed Networking** submitted by Władysław Olesiński in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**

. . . .
Prof. Paweł Gburzyński

. . . .
Prof. Carl McCrosky

. . . .
Prof. Jacek Tuszynski

. . . . J. Harms . . . .
Prof. Janelle Harms

. . . .
Prof. Ursula Maydell

Date: . . Nov 17, 1998 .

To Madzia

Abstract

Rapid progress in computer technology, processor speed and memory capacity, as well as demand for digital audio and video pose new problems in communication networks. To accommodate a whole range of new applications, the networks must operate at very high speeds, and they often require new protocols. These networks are supposed to employ the connection-oriented paradigm as the only solution that will achieve the desired very high speeds. However, this paradigm, although it does possess certain virtues, is not free from disadvantages that make the effective implementation of such networks difficult.

In this dissertation, we study the suitability of a connection-less paradigm in high-speed networks. As an example of this paradigm, we consider *deflection routing*. This scheme is relatively new and in many ways it is alternative to existing solutions. We show that deflection routing could be used as an underlying technology allowing to satisfy the requirements posed by high-speed applications.

High-speed isochronous applications, like video, require small deviations from the average interarrival delay (low jitter). We show that deflection networks, equipped with reassembly buffers of modest size, can smooth out the jitter and assure a very low packet loss regardless of the traffic pattern and network topology.

Another important feature required by high-speed applications is the ability to send a packet to a number of distinct destinations (multicast). Traditional techniques of multicasting are not applicable in deflection networks. We propose and study several simple multicast schemes providing deflection networks with this important feature.

Having shown that jitter may be contained and that multicasting in deflection networks is possible, we study the performance of several video applications, including those that require multicasting. We show that in a realistic environment, performance of these applications in deflection networks is very good.

In addition to low jitter and multicasting, quality of service is another requirement that must be fulfilled by a high-speed network. We propose and investigate a few simple means for sustaining the given throughput of an isochronous stream regardless of the intensity and

pattern of the background datagram traffic. We also study the effect that granting priorities to isochronous traffic has on the throughput and jitter.

However, we notice that these measures fall short of providing long term quality of service in cases in which, e.g., there is a very large number of isochronous sources that contribute some traffic to the network. We propose a protocol that allows a source to determine whether its session may be accepted by the network. At the same time, possible variations of the interarrival delay caused by the sources trying to initiate their sessions, are easily smoothed out by the receivers that can dynamically adjust their playout buffers. We show, with the example of a high-speed application, that this protocol makes it possible to achieve a required long term quality of service.

Slotted deflection networks, which so far have been considered by most researchers as the prevailing implementation of deflection, may be difficult to implement in high-speed environments because of problems with synchronization. Therefore, we study asynchronous deflection as a more realistic alternative. In general, asynchronous deflection networks offer lower throughput than their synchronous counterparts — this is the price paid for a more feasible implementation and a less complex routing. We show how transient buffers affect the throughput of an asynchronous network. We also show that the appropriate size of these buffers may greatly improve the throughput of an asynchronous deflection network, making it close to the throughput achievable in a synchronous network.

Finally, we compare the performance of a deflection network with the performance of a store-and-forward network. We show that if no resources are reserved in advance, the performance achieved by the deflection network significantly exceeds the performance of a store-and-forward network. At the same time, buffer space requirements in the latter one are much higher.

Acknowledgements

I wish to express my sincere gratitude to my supervisor Professor Paweł Gburzyński for always being willing and available to offer me his guidance, advice and encouragement. I also thank him for his support during my first months in Edmonton.

I would also like to thank the members of my committee Professor Carl McCrosky, Professor Jacek Tuszynski, and especially Professor Janelle Harms and Professor Ursula Maydell for their valuable comments.

I want to thank Dr. Catherine Desheneau for her appreciation of my work as a teaching assistant.

My wife deserves special thanks. Her love, support and companionship tremendously helped me during my graduate studies. With her, every success is more enjoyable, and every problem more bearable.

I am indebted to my parents for their encouragement to pursue my interests, and especially my mother for all her love and attention.

Many thanks to my brothers Włodek, Rysiu and Jasiu for their constant support and encouragement. Especially, thanks to Włodek for giving me the idea to study computing science at the University of Alberta, and Jasiu, for being a great older brother in my childhood.

Finally, thanks are due to Paweł for his friendship and all the good times, as well as to his parents Mr. and Mrs. Jachowicz for making me and my wife feel welcome at their home and for introducing us to cross-country skiing.

Contents

1 Introduction **1**

1.1 Challenges of High-speed Networks 1

1.2 Overview of High-speed Implementations 6

1.2.1 SONET and SDH 6

1.2.2 WDM Networks 6

1.2.3 HIPPI 7

1.2.4 Gigabit Ethernet 7

1.2.5 CBR 8

1.2.6 DQDB 8

1.2.7 CSMA/RN 8

1.3 Deflection Networks—Connection-less vs Connection-oriented Paradigm 9

1.4 Outline of the Dissertation 15

2 Real-time Traffic in Deflection Networks **17**

2.1 Network Model 17

2.2 Results 19

2.2.1 Poisson Background Traffic 19

2.2.2 Correlated Poisson Background Traffic 20

2.2.3 Bursty Synchronous Background Traffic 22

2.2.4 Other Traffic Patterns 24

2.2.5 Larger Propagation Delays 25

2.2.6 Reassembly Buffers 30

2.3 Summary 30

3 Multicast in Deflection Networks **32**

3.1 Network Model 33

3.2 Basic Multicast Schemes 34
3.3 Implementation 35
3.4 Results 37
    3.4.1 Basic Schemes 37
    3.4.2 Limited Replication Schemes 43
3.5 Comparison Between the Basic and Limited Replication Schemes 44
3.6 Extended Replication Schemes 48
    3.6.1 Extended Replication 48
    3.6.2 Class Replication 49
    3.6.3 Comparison of the Extended Schemes 50
3.7 Multicast Groups 53
3.8 Summary 53

4 Video Applications in Deflection Networks **55**
4.1 Network Model 55
4.2 Results 58
    4.2.1 Videophone Traffic 59
    4.2.2 Videoconference Traffic 64
    4.2.3 Video Movie Traffic 66
4.3 Summary 70

5 Techniques Giving a Preferred Treatment to Isochronous Traffic **73**
5.1 Network Model 74
5.2 Prioritized Routing for Synchronous Traffic 75
5.3 Throttling Datagram Traffic 79
5.4 Results 82
    5.4.1 Poisson Datagram Traffic 83
    5.4.2 Correlated Poisson Datagram Traffic 86
    5.4.3 Bursty Synchronous Datagram Traffic 87
5.5 Summary 88

6 Service Guarantees in Deflection Networks **91**
6.1 Network Model 92
6.2 The Protocol 93
    6.2.1 Active Phase 93

6.2.2 Setup Phase 97
6.2.3 The Receiver 100
6.3 Results 103
6.4 Possible Improvements of the Protocol 110
6.5 Summary 112

7 Asynchronous Deflection with Transient Buffers **115**
7.1 Network Model 116
7.2 The Standard Algorithm 119
7.2.1 Performance of the Standard Algorithm 119
7.2.2 Standard Algorithm with Longer Delay Lines 124
7.2.3 Variable Packet Length 125
7.3 The "Quick" Algorithm 126
7.3.1 Comparison with the Standard Algorithm 127
7.3.2 Longer Delay Lines 128
7.4 The Complete Algorithm 129
7.5 Comparison of Asynchronous and Synchronous Routing Algorithms 132
7.6 Summary 133

8 Deflection Networks vs Store-and-Forward Networks **135**
8.1 Network Model 135
8.1.1 Deflection Network 135
8.1.2 Store-and-forward Network 135
8.2 Results 137
8.2.1 Videophone Traffic 137
8.2.2 Transmission of Video Movies 141
8.3 Summary 143

9 Conclusions and Suggestions for Future Research **144**
9.1 Conclusions 144
9.2 Future Research 146
9.2.1 Distributed Computing 146
9.2.2 Mobile Computing 146
9.2.3 Comparison with Other Networks 147
9.2.4 Further Studies of the Quality of Service 147

Bibliography **148**

A Control messages **152**

B The Simulator **153**

B.1 Slotted Simulator 153

B.2 Asynchronous Simulator 155

List of Tables

7.1 Maximum and saturated throughput for different W and k (standard algorithm) 119

7.2 Standard algorithm, load 14, k=2 121

7.3 Standard algorithm, load 32, k=4 123

7.4 Maximum and saturated throughput for different F, W, and k (standard algorithm with longer delays) 124

7.5 Standard algorithm with longer delay lines, load 14, W=1024, k=2 125

7.6 Standard algorithm with longer delay lines, load 32, W=256, k=4 125

7.7 Maximum and saturated throughput for different W and k (variable packet length) 125

7.8 Standard algorithm (variable packet length), load 14, k=2 126

7.9 Standard algorithm (variable packet length), load 32, k=4 126

7.10 Maximum and saturated throughput of standard (S) and quick (Q) algorithms 127

7.11 Quick algorithm, load 20, k=2 127

7.12 Quick algorithm, load 50, k=4 128

7.13 Maximum and saturated throughput for different F, W, and k (quick algorithm with longer delay lines) 128

7.14 Quick algorithm with different F, load 20, $W = 1024$, $k = 2$ 129

7.15 Quick algorithm with different F, load 50, $W = 256$, $k = 4$ 129

7.16 Maximum and saturated throughput of the standard (S) and complete (C) algorithms for different W and k 130

7.17 Complete algorithm, load 14, k=2 132

7.18 Complete algorithm, load 32, k=4 132

7.19 Maximum and saturated throughput for different W and k—comparison of asynchronous schemes (S-standard, Q-Quick, C-Complete) 133

7.20 Maximum and saturated throughput for different W and k—comparison with synchronous routing 134

8.1 Deflection network in videophone application (k=2) 138
8.2 Store-and-forward network in videophone application (k=2) 138
8.3 Deflection network in videophone application (k=4) 140
8.4 Store-and-forward network in videophone application (k=4) 140
8.5 Deflection network in transmission of video movies (k=2) 142
8.6 Store-and-forward network in transmission of video movies (k=2) 142
8.7 Deflection network in transmission of video movies (k=4) 142
8.8 Store-and-forward network in transmission of video movies (k=4) 143

List of Figures

1.1 Network topologies 11

2.1 Ring topology, size 8, connectivity 4 18
2.2 Uniform Poisson background traffic in torus and ring networks 21
2.3 Uniform Poisson background traffic in triangle network 22
2.4 Correlated Poisson background traffic in torus and ring networks of: $P = 0.4$, rate of S is $1/14$, $n = 51$ (in larger network) or $n = 12$ (in smaller network) 23
2.5 Correlated Poisson background traffic in triangle network: $P = 0.4$, rate of S is $1/14$, $n = 51$ (in larger network) or $n = 12$ (in smaller network) 24
2.6 Bursty synchronous background traffic in networks of all three topologies and all sizes ($t = 20$) 25
2.7 Uniform Poisson background traffic for different propagation delays in large networks of different topologies (rate of S is $1/15$) 27
2.8 Uniform Poisson background traffic in the large networks with links of random lengths 29

3.1 Basic schemes (no replication), connectivity 2 37
3.2 Number of bounces vs number of multicast destinations and vs background load. Basic schemes, connectivity 2 38
3.3 Delay vs number D of multicast destinations and vs background load. Basic schemes, connectivity 4 40
3.4 Number of bounces vs number of multicast destinations and vs background load. Basic schemes, connectivity 4 40
3.5 Average length of the path with basic schemes (no replication), connectivity 2 and 4 42
3.6 Delay vs number D of multicast destinations and background load. Replication schemes, connectivity 2 44

3.7 Delay vs number D of multicast destinations and background load. Replication schemes, connectivity 4 45

3.8 Delay vs number D of multicast destinations and vs background load. Comparison of the schemes, connectivity 2 45

3.9 Delay vs number D of multicast destinations and vs background load. Comparison between the schemes, connectivity 4 46

3.10 Throughput and delay vs number D of multicast destinations. Comparison between R_Random and point-to-point scheme 47

3.11 A fragment of a 16x16 torus network 49

3.12 Delay vs number D for different replication distances R_d in R_Ext, and in R_Class 50

3.13 Delay vs background load for different replication distances R_d in R_Ext, and in R_Class 51

4.1 Videophone synthetic trace 59

4.2 Average packet loss and average end-to-end delay for both connectivities . . 60

4.3 Average packet loss and end-to-end delay—comparison of synchronized and asynchronized versions of the algorithm 63

4.4 Average packet loss and end-to-end del_y in videoconference 65

4.5 Video movie synthetic trace 67

4.6 Average packet loss and average end-to-end delay in transmission of video movies 68

5.1 Throughput under uniform Poisson datagram traffic for different probabilities P, datagram traffic contribution 0.16, and rate of stream sources 1/10 . . . 76

5.2 Number of blocked and aborted calls under uniform Poisson datagram traffic for different probabilities P, datagram traffic contribution 0.16, and rate of stream sources 1/10 77

5.3 Jitter under uniform Poisson datagram traffic for different probabilities P, datagram traffic contribution 0.16, and rate of stream sources 1/10 78

5.4 Throughput and jitter under uniform Poisson datagram traffic for different contributions c of stream traffic and transmission rate of stream sources is 1/10 84

5.5 Throughput under uniform Poisson datagram traffic for contribution $c = 0.12$ of stream traffic and transmission rate of stream sources 1/10 85

5.6 Throughput under correlated Poisson datagram traffic for different contributions c of stream traffic, transmission rate of stream sources 1/10, $P = 0.4$, and frequency of pair changes $l = 1000$ 86

5.7 Throughput under correlated Poisson datagram traffic for contribution $c = 0.12$ of stream traffic, transmission rate of stream sources 1/10, $P = 0.4$, and frequency of pair changes $l = 1000$ 87

5.8 Throughput under bursty datagram traffic for different contributions c of stream traffic, transmission rate of stream sources 1/10, and $t = 20$ 88

5.9 Throughput under bursty datagram traffic for contribution $c = 0.12$ of stream traffic, transmission rate of stream sources 1/10, $t = 20$ 89

6.1 Separating of transmission rates 96

6.2 Number of active and blocked pairs vs number of activated pairs 106

6.3 Packet loss vs number of pairs in the 2- and 4-connected network 107

6.4 End-to-end delay and throughput vs number of pairs in the 2- and 4-connected network in all three environments 109

7.1 Simplified model of a switch 117

7.2 Input buffer 117

7.3 Maximum overall throughput vs waiting time W in the buffer for different connectivities 120

7.4 Standard algorithm, $k = 2$ 120

7.5 Standard algorithm, $k = 4$ 123

7.6 Saturated throughput vs F for different connectivities 124

7.7 Quick algorithm with longer delay lines, $k = 2$ 128

7.8 Complete algorithm, $k = 4$ 131

7.9 Complete algorithm, $k = 2$ 131

Chapter 1

Introduction

Connection-less and connection-oriented paradigms define the operation of the network layer. If the notion of a circuit or a stream appears in the network layer, we say that the network operates according to the connection-oriented paradigm. If packets are processed (routed) individually, we say that the paradigm of the network layer is connection-less.

Both paradigms have found wide applications in networks. The connection-less paradigm is predominant in Local Area Networks (LANs) while the connection-oriented paradigm is used where a reliable, ordered stream of packets is expected. The latter one, exemplified by the Asynchronous Transfer Mode (ATM) is commonly viewed as the only viable solution for very high speed networks.

It is a common presumption that connection-less networks are not suitable for high-speed environments because datagrams are less reliable and, since they may arrive to destinations misordered, they need to be ordered, which further decreases the chance of their usability. In this thesis, we study the connection-less paradigm in the context of high-speed networks. We show that this paradigm, exemplified by deflection networks, is suitable in high-speed environments.

In this chapter, we describe some challenges posed by high-speed networks. Then, we present an overview of several gigabit implementations, introduce deflection networks, and give the outline of the dissertation.

1.1 Challenges of High-speed Networks

By very high-speed networks we mean networks with capacities of gigabits per second. Building high-speed networks is difficult. The tremendous rate at which packets arrive at the switch and the multitude of functions that must be performed before the packet is forwarded onto the outgoing port, is an obstacle in building very high-speed networks.

High speeds also have an impact on various functions performed by networks such as admission or congestion control. Below, we take a closer look at some of the challenges that come with the transmission of data at gigabit speeds.

Since transmission rates in high-speed networks are much higher than in conventional networks, a large number of packets may be in transit in the network at any given moment. That is why the destination cannot effectively control the source. By the time a message from the destination arrives at the source, a large number of new packets may be sent possibly increasing the congestion.

Generally, traditional traffic models may not be valid in the context of data networks because of the assumption that packets arriving at a switch are independent. They may work well in telephone networks (where they represent calls) but not in the packet data networks. Jain and Routhier [JR86] have shown that packets tend to travel in so called *packet trains*. About 30% of packets in the network may have the same destination as the previous packet. Mogul [Mog91] has shown that about 70% of the segments received by UDP and TCP hosts are destined for the same port as their predecessors. Such packet trains (or just correlated traffic) can be found even in data networks highly loaded with traffic from a large number of sources. This clearly shows that traffic in data networks is hardly independent.

Also traffic in such networks is much more bursty [KS92, Kun92]. It was shown in [LTWW94] that it has self-similar (fractal) properties rather than Poisson. This again indicates that traditional traffic models commonly used in telephone networks do not work in data networks. For example, statistical multiplexing schemes rely strongly on the law of large numbers. That is, there is an assumption that the total amount of bandwidth required to satisfy all streams stays nearly constant, even if the individual flows vary the amount of traffic they send. This suggests that the aggregated traffic coming from many sources will have a tendency to be "smooth". This assumption was proven to be false. Actually, the aggregated traffic tends to be as bursty as the contributing streams. The increase in traffic burstiness is also due to the widespread use of distributed applications and high speed computers capable of transmitting large amounts of data in a very short time.

These phenomena may cause congestion, not easily controllable in conventional ways. The following methods (see [Kun92]) may become obsolete (or at least will require some serious changes):

- Statistical load prediction.

This method works well when there are many low-speed traffic streams. In high speed networks, the variation in the network load caused by just a few traffic sources can increase significantly. This makes useful network load predictions almost impossible. Moreover, as we have mentioned above, traditional Poisson models that could be used in predicting the traffic characteristics may not work.

- Overdesigned network capacity.
  This approach works for constant bit-rate traffic at moderate speeds. However, for a highly bursty traffic produced by very high-bandwidth traffic sources, it would be impractical to overdesign networks. If the network is highly over-designed to accommodate peak traffic, the resources are wasted if the current traffic is moderate. On the other hand, if the network is not over-designed, packet loss will be high due to the bursty nature of the traffic.

- Admission and congestion control.
  To avoid overloading the network, one can apply admission control to traffic sources. A session can be established only if the admission control algorithm determines that all the required network resources will be available to handle the load associated with the sessions. Without a fast feedback from the network to the traffic source, the traffic source cannot be notified in time about the possible congestion or releases of network resources by other traffic. Fast feedback may be difficult in high-speed networks because network overflow can occur within a short time. Commonly used feedback methods, in which congestion is assumed if the protocol discovers lost packets or experiences increased round-trip delays, may no longer be appropriate. It may be simply too late to take action to avoid the present congestion since many new packets may have entered the network in the time that elapsed between the moment when the congestion began, and the moment when the network starts to respond.

- Network buffering.
  In this approach, buffers are placed at various switches in the network to hold excessive traffic. However, buffers in high-speed networks may be filled quickly, which may increase packet loss. Once the traffic has been buffered, the network is committed to handle it somehow—the buffered traffic must be eventually cleared from the network, which requires time. The queueing delays can get so long that by the time the packets come out of the switch, most of them may have been already retransmitted due to timeouts. This, of course, wastes network resources. Equipping the switches with

large buffers would cause delays and high jitter which is undesirable, e.g., in video and voice applications.

It seems that in high-speed networks, flow-control mechanisms should be more preventive than reactive because, in the latter case, the involved parties must exchange status information across the large normalized diameter of the network.

In one approach, the user must specify the quality of service demanded from the network. This includes e.g., the peak rate, and the average and maximum burst size. After the session has been admitted to the network, the protocol must monitor the user's adherence to the declared parameters. This process, due to the statistical and unpredictable nature of data flow, is difficult. Rate-based mechanisms control the negotiated transmission rate. Leaky bucket [Tur86] is a well known example of this technique. With this solution, a virtual bucket is filled by incoming packets and is emptied at the negotiated, constant rate. Packets arriving when the bucket is full are discarded.

Token access systems are another alternative. In particular, they are widely used in MAN networks (e.g., FDDI [Ros89]). Unfortunately, they scale poorly to high speeds [DG93, DG94] because during the periods in which the token is "in transit", the bandwidth is wasted. These periods must be subtracted from the throughput achievable in the network. With the increasing network's transmission rate, the impact of the token transition time becomes more significant and the throughput of the network becomes a smaller and smaller fraction of the channel capacity.

More specifically, let L be the round-trip propagation delay of the ring (expressed in bits), l be the average length of a packet, and $a = L/l$ [DG93]. The value of a tells how many packets can be inserted to the link before the first bit of the first packet appears at the other end. Large a may result from a high transmission rate of the network or a very long length of a ring. The performance of protocols like FDDI deteriorates when a significantly exceeds 1. For example, the throughput achievable by FDDI can be expressed as

$$T = \frac{\Sigma THT}{\Sigma THT + L}$$

where ΣTHT is the sum of the token holding times at all stations (and can be viewed as the packet length l). We can see that the throughput is inversely proportional to L. Thus, if FDDI is scaled to higher and higher speeds, the effective throughput becomes a smaller and smaller fraction of the channel capacity. Also, it does not grow when more stations are added to the network. The more stations that network supports, the smaller fraction of T is available to one station.

For the same reason, shared media networks (e.g., Ethernet [MB76]) do not scale well. The maximum throughput of Ethernet, for the optimal case of ϵ contention slots per frame, was calculated by [Tan96] as

$$T = \frac{1}{1 + 2CL\epsilon/cF}$$

where F is the packet length, C is the network bandwidth, L is the cable length, and c is the speed of signal propagation. Clearly, when the CL product increases, that is, when the network's speed and/or size increases, the throughput for the given packet size drops.

When the network is scaled up to higher speeds, the propagation delay in a medium stays constant but many more bits may be packed into the medium. Thus, access techniques whose performance is affected by the length of the network (or the number of bits that can be in the network at any time) are bound to exhibit low normalized throughput.

Notably, deflection networks presented in the next section, do not posses this unpleasant property. For example, the throughput of the 2-connected Manhattan Street Network (MSN) is limited by $4n$, where n^2 is the size of the network [GG86, BDG95]. We can see that the throughput in MSN is independent of the propagation delay or capacity. It actually grows with the growing number of transmission sources.

In order to function in a high-speed environment, a network must support the following:

- Possibility of achieving gigabit or near gigabit speeds.

  This means that the design of the network and the protocol itself should be simple enough to make such speeds possible.

- Mechanisms to provide at least rough bounds on the end-to-end delay [Par94].

  This is particularly important in isochronous applications (voice and video). The studies show that for example end-to-end delays from the audio equipment attached to the network should not exceed 600 ms [RK63]. This concerns not only traditional phone but new applications like videophone.

- Predictable, modest jitter.

  Some applications like real-time voice and video streams are isochronous in nature i.e., they generate streams of finite samples which are transmitted and received at fixed time intervals. Transmission errors, congestion, and other failures cause delay variations (jitter). That is why buffers are used to smooth out the jitter before presenting the packets to the upper level of the protocol stack. Rough estimates of jitter should be known so that the appropriate buffer size could be allocated—buffer size that could assure both a tolerable packet loss and end-to-end delay.

- Multicast capability.

  This is important, particularly in applications like transmission of video movies or videoconferencing. A source must be able to address a single packet to a number of receivers. This greatly saves bandwidth in the network.

- Quality of service.

  Obviously, the quality of a connection demanded by the user should be sustained throughout the session, regardless of other activities in the network. If the given quality cannot be provided, the user should be notified about it and a session should be (preferably) blocked. However, once the session has been admitted to the network, its quality should not deteriorate.

1.2 Overview of High-speed Implementations

We saw in the previous section that scaling the networks to gigabit speeds poses some new problems that often stem from that fact that the relative length of the network (expressed in bits) is large. However, there exist several technologies, of which some are still rather experimental, that support gigabit speeds. Below, we present a short overview of several solutions.

1.2.1 SONET and SDH

SONET (Synchronous Optical Network), and its European counterpart SDH (Synchronous Digital Hierarchy), is part of a telephony standard developed to support multiplexing on links capable of data rates of hundreds of megabits per second. Its goal is to develop a set of multiplexing standards for high-speed links. The rates range from 51.84 Mb/s to about 2.5 Gb/s.

With this approach, copper wires are replaced by multiplexed fiber links using SONET.

1.2.2 WDM Networks

Wavelength Division Multiplexing (WDM) networks use the special properties of optical fibers to build data networks. The bandwidth of a fiber is divided into multiple channels. These channels are then arranged such that particular hosts can communicate with each other. There are two types of WDM networks: single-hop and multihop. In single-hop WDM networks, every node can communicate directly with all other nodes. In multihop WDM networks, a node can communicate directly with only a few other nodes.

Examples of this technology include LAMBDANET [GKV$^+$90]. In this network consisting of n nodes, each node has a single, fixed transmitter and n receivers, one for each sending channel. The receiver listens to all channels and selects the relevant transmission. Multicasting in LAMBDANET is easy, and there is no need to schedule transmissions (everyone has its own channel). However, every node must have n receivers which becomes very expensive with increasing n.

This approach differs from SONET in that it splits large bandwidth of a fiber into multiple channels, each channel shared by the subset of users.

1.2.3 HIPPI

High-Performance Parallel Interface (HIPPI) is a point-to-point connection technology, designed to connect two devices at speeds up to 1600 Mb/s [TR93]. It was the first standard way to connect devices at high data rates.

However, HIPPI does not support multicasting and the cable length in standard HIPPI cannot exceed 25 meters.

1.2.4 Gigabit Ethernet

Gigabit Ethernet is a candidate for very high-speed LANs. Inherently, Ethernet is difficult to scale up to higher speeds. In standard, 10Mb/s Ethernet, stations can be located no more than 2 kilometers apart. This distance limitation results from the relation between the time required to transmit a minimum-sized Ethernet packet (64 bytes) and the round-trip propagation delay. To keep the same minimum size of a packet at 1Gb/s speed, the diameter of the network would have to shrink from 2 kilometers to 20 meters, which is unacceptable. For this reason, the minimum size of the packet in Gigabit Ethernet is increased to 512 bytes, the maximum bus length is 200 meters (like in 100Mb/s Fast Ethernet), and numerous safety margins built into engineering specifications are eliminated (to make a combination of the above packet size and the bus length possible). However, several problems still remain. For example, when the packet size is less than 512, some bandwidth is wasted because the short packets must be padded—at the same time, the average packet size on most Ethernets is between 200 and 500 bytes. Also, Gigabit Ethernet does not provide the quality of service required by multimedia applications.

1.2.5 CBR

Cambridge Backbone Ring [GLH90] is a ring network divided into a number of frames which rotate around the network. Each frame contains four slots, and each slot contains one cell. To send data, a station looks for an empty slot by scanning each frame. When it finds an empty slot, the station inserts the data and marks the slot as busy. When a slot (in a frame) arrives to the destination, the destination copies the data from the slot. Slot may be reused only when it arrives back to the sender.

CBR supports multicasting and, to some extent, performance guarantees (the maximum delay is bounded). However, allowing a station to use multiple slots can cause super-token behaviour. That is, when one station sends many packets onto the otherwise inactive network, downstream stations cannot send their packets and their queues accumulate packets. When finally the station empties its queue, its downstream neighbour begins transmission of its packets, blocking all other stations. Thus, each stations gets access to the network in big clumps, and the network acts as if it used a single token to control the access rights.

To avoid the super-token behaviour, CBR does not permit stations to fill successive slots. Also, the station that frees a slot cannot reuse it but must pass it on. However, under the light load, this approach wastes some bandwidth. Bandwidth is also wasted because only the sending stations can clear the slots.

1.2.6 DQDB

In the Distributed Queue Dual Bus [DQD91], every station is connected to two unidirectional busses. The two busses transmit in opposite directions, so there exists a path from every station to every other station. To send a cell, a system must reserve a slot on the appropriate bus. DQDB places cells in individual slots. To support traffic with real-time delivery requirements, DQDB allows slots to be permanently reserved.

Disadvantages of DQDB include problems with fairness. Also, it was shown in [MK93] that in many aspects, DQDB is inferior to deflection networks, presented in the next section.

1.2.7 CSMA/RN

Carrier Sense Multiple Access/Ring Network is an interesting protocol proposed in [FMO$^+$91]. Data from the ring enters the station through a delay buffer. If a packet on the ring is addressed to the given station, the station removes it from the ring. A station can transmit its own packet only if it perceives no activity in the delay buffer. However, while the packet is being transmitted, a packet in transit from the ring may enter the delay buffer. In that case, the station interrupts the transmission and puts a truncation marker on the end of its packet. This indicates that the packet is incomplete and the rest of it will be sent later.

Packets in transit always have priorities over new packets, so the maximum delay in CSMA/RN is bounded. However, when the load is high, a packet may be divided into many small fragments, which will waste some bandwidth with their headers and truncation markers. Also, there is no obvious way to provide a quality of service.

1.3 Deflection Networks—Connection-less vs Connection-oriented Paradigm

By a *pure deflection network* we understand a switching network that never loses a packet at an intermediate switch because of a limited buffer space. Packets that cannot be relayed via their preferred routes (because those routes are busy) are *deflected* via suboptimal routes. This concept is similar to "hot-potato routing" [Bar64]. Deflection networks are traditionally illustrated by the Manhattan-street networks proposed by Maxemchuk in [Max85, Max87]. He also studied the effect of buffering at the nodes on the performance [Max89]. The analytical models of deflection were presented in [GG86, Cho91, GM93]. Deflection routing in hypercube networks was studied in [GH92], while the performance of deflection routing in shuffle-based networks was studied in [KH90]. Deflection routing has also been studied in [Max89, Max91, CM91, MK93, HRL95, DG96]. Asynchronous deflection networks were studied in [MWW90a, MWW90b, GM93].

In the previous section, we mentioned several problems that come with buffering. In deflection networks, no buffer space is required for storing packets that cannot be immediately relayed via their optimal routes, although it may make sense to store such packets temporarily before deflecting them right away [Max87]. Elimination of buffers can speed up switching significantly, particularly if purely photonic switching becomes feasible.

Because of deflections, packets belonging to the same session may arrive at the destination out of order. This property of deflection networks has been traditionally perceived as a serious disadvantage compromising their reliability in applications in which the timing of arriving packets is important (e.g., voice and video). This presumption has brought us the connection-oriented paradigm of ATM as the only solution to be adopted by the "serious" networks of the future. But, as the authors of [BDG95] have argued, connection-oriented networks are haunted by other problems that are absent in deflection networks. The proliferation of traffic classes in ATM and the inefficient ways in which ATM handles datagram traffic [LB92], indicate that the connection-oriented approach is not perfect and that it hardly solves the problems of datagram-oriented networks (including deflection networks).

Because of the statistical multiplexing and unpredictability of real-life traffic scenarios, switched networks are bound to lose packets. In a deflection network, the problem of packet misordering is solved at the destinations by using *reassembly buffers* [CM91, HRL95]. A packet can be lost, if it happens to arrive so late that the reassembly buffer can no longer help. Thus we are confronting two approaches: one in which packets can be lost at any intermediate switch-relay because of the limited buffer space, and another in which packets can be lost at the destination for essentially the same reason. Thus, at this level of perception, there is no significant difference between the two approaches.

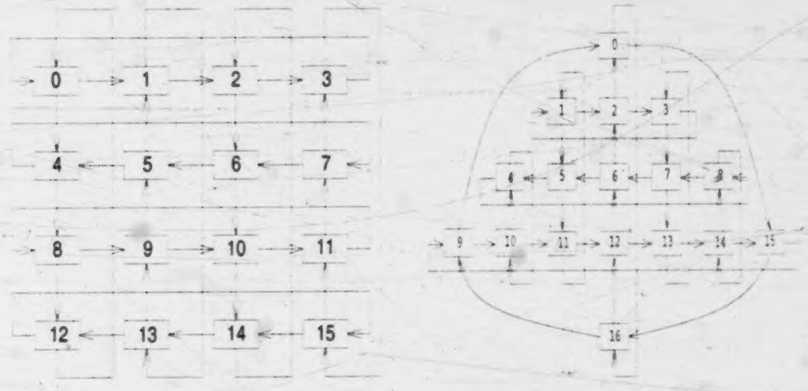
One advantage of deflection networks over store-and-forward networks is that the former do not try to "fix what ain't broke." In particular, datagram traffic requires no reassembly buffer at the destination (and no buffer space anywhere in the network). A destination in a deflection network allocates the reassembly space on a per session basis; thus, buffers are only used when required by the session. Clearly, the destination can "know better" how much buffer space is needed to accommodate its needs. Even if at first sight the amount of buffer space needed to accommodate an isochronous session appears somewhat large, one should keep in mind the simplicity and locality of the underlying allocation problems.

If we give up connection-orientedness as the prerequisite for networking, we will see that the number of sessions that actually need this paradigm will drop significantly. Let us point out that many traffic scenarios traditionally viewed as stream-oriented are only so because some old-fashioned protocols insist on viewing them that way. File transfer (FTP) is a good example. Note that the operating systems of the hosts involved in a file transfer perceive the file as a random collection of pages, which can be transferred independently in any order. Even better, the source could actually optimize the transfer by selecting the pages in the order that would minimize the total time needed to read them from the disk.

If we look carefully at those communication scenarios that appear to require the preservation of packet ordering, we will see that most of them fit into the following categories:

- Scenarios that could be carried out with packets arriving in any order. They enforce packet ordering because some obsolete higher-level protocols view them as stream-oriented scenarios.

- Scenarios involving relatively short transfers (e.g., a piece of text to appear on a screen). Messages of this sort can be safely reassembled in a small buffer space at the destination.



(a) Torus, size 16, connectivity 2

(b) Triangle, size 17, connectivity 2

Figure 1.1: Network topologies

- Long, sustained, isochronous transfers that actually require packets to arrive in order (e.g., video, voice).

Note that the scenarios from the last category typically admit a non-zero packet loss rate. Consequently, one can implement them with a limited reassembly buffer, dropping packets that arrive out of sequence while the buffer is full.

Similar to ATM networks, deflection networks are open for simple schemes aimed at improving the quality of service for some traffic patterns, based on traffic shaping at the source. Many people believe that in the face of long network delays and traffic unpredictability at intermediate switches, source policing is the only viable solution to the problems of congestion in statistically multiplexing networks.

In this thesis, specifically in our experiments, we consider mainly two deflection network topologies: torus, and triangle (Figure 1.1). The torus represents regular networks with good reachability. The triangle is a topologically biased network: different switches have a different perception of their neighbourhood. In Chapter 2 we also consider ring which is still regular, but its reachability is poorer than in the torus.

We assume that each switch is equipped with delay buffers whose purpose is not to store packets before they are forwarded, but to align packets arriving at the switch and to give the switch ample time to make a routing decision. The networks operate in a slotted manner, in a way similar to MSN [Max87].[1] In one routing cycle, the switch accepts incoming slots from all its input ports and routes them to the output ports. If an incoming slot is nonempty and addressed to the current switch, the switch receives the contents of the slot and marks it as empty. If an incoming slot is empty, the switch is free to fill it with its own outgoing packet. The routing decision assigns output ports to all incoming slots that appear nonempty after the above operations.

---

[1] Note that the 2-connected torus is in fact a Manhattan Street Network.

There are a number of deflection routing strategies that decide which packet should be deflected. In the simplest, *standard* one, a fair coin toss decides which packet is deflected..

In the *age* routing strategy, the packet traveling longer (e.g., having traveled the greater number of hops) is given priority over the other packet.

A *distance* routing strategy assigns higher priority to packets that are closer to the destination.

The best (in most cases) routing strategy, is the one called *locally optimal* [BC90]. Let $d(S_1, S_2)$ denote the length of the shortest path from switch S_1 to S_2, expressed as the number of links (hops). Let $S(s)$ denote the destination switch of a nonempty slot s. Assume that switch S_a is making a routing decision for k nonempty slots routed in the current cycle. This decision is carried out in such a way as to minimize $P = \sum_{i=0}^{k-1} d(S_i, S(s_i))$, where s_i is a nonempty slot being assigned to an output port, and S_i is the immediate neighbour of S_a to which the slot is routed. If several assignments of the outgoing slots to output ports produce the same minimum value of P, one of them is chosen at random. In other words, packets are assigned to outgoing links such that the sum of the distances to their destinations is minimized. The randomization of the routing rules was postulated in [Max91] for MSN as a means of avoiding *livelocks*. Comparison of various deflection strategies can be found in [HRL95].

We assume that slots are never buffered at a switch, except for the alignment and routing. Although generally a higher throughput can be achieved by using limited buffers, it was shown in [GG86] that for MSN

$$\lim_{N\to\infty} T(B)/C = 1$$

where N is the number of switches in the network, B is the amount of extra buffer space at a switch, $T(B)$ is the observed throughput with the extra buffer space, and C is the network capacity, i.e., the maximum throughput reachable with infinite buffers. This spectacular result holds for any B, including 0. It is suspected that the same formula holds for all deflection networks with a reasonable topology and connectivity [BF92]. Thus the impact of the extra buffer space on the network performance asymptotically vanishes as the network becomes larger and larger.

Now, let us analyze how deflection networks could solve some of the problems posed by high speed environments. If we deal with very high-speed networks, both the electronics of a switch and the routing strategy should be simple. Switches in contemporary networks tend to be very complex. For example, crossbar switches like Knockout ([YHA87]) or Gauss require circuitry whose complexity grows as n^2 where n is the number of ports. Moreover, they do not eliminate packet loss caused by overflowing output buffers. Their performance further deteriorates because packet arrivals in data traffic are related. Some studies show [GHM$^+$91] that to maintain a given packet loss, the switches may require ten times more buffering than predicted by the M/D/1 model. This once again indicates that traditional queueing models are not adequate in high-speed networks. The task of routing packets (or cells) with gigabit speeds in such switches is rather difficult.

To see better the complexity of routing in a high-speed network, let us assume a 53 byte cell of ATM network and the transmission rate of 1 Gb/s. Under such conditions, a switch has 424 ns to route a packet, that is, select the appropriate output link on which the packet should be relayed, and resolve a possible contention in such a way that the network performance is optimized.

In deflection networks, there are only a few incoming and outgoing links. Routing decisions are made solely on the basis of information extracted from the packet's header (i.e., destination's address). This makes costly exchanges of routing tables between the switches unnecessary.

Also, there is no buffer space to store packets that cannot be routed over their preferred ports. Thus, purely photonic switching becomes feasible, and electronic-to-optic and optic-to-electronic conversions can be avoided.

The complexity of routing in deflection networks may be further reduced if we opt for asynchronous routing (studied in Chapter 7). With this approach, a packet that arrives at a switch is relayed on the most suitable from the currently available outgoing links. Also, there is no need for slot alignment. This approach is particularly attractive when the network connectivity is high. This all greatly simplifies the implementation of a switch and significantly improves its ability to route packets with gigabit speeds.

As we have mentioned above, the fact that no intermediate buffers are needed in switches, except for those that give them sufficient time to make routing decisions, offers one more advantage. Namely, packets are never dropped at intermediate switches even under extremely high load.

Flow control is also easily solved in deflection networks. Since new packets may be inserted to the network only if there are enough free slots available, a heavily loaded network with a shortage of free slots, simply does not allow the sources to transmit and thereby throttles the excessive traffic. In this way, the network behaviour tends to be stable under any load, i.e., the throughput cannot collapse due to the saturation of some resources.

Deflection networks are usually considered in the context of mesh topologies. Such topologies have several interesting features.

- *Deflection index*, that is, the number of hops a single deflection adds to the packet's delay in the 2-connected network is always 4, regardless of the network size.
- A high percentage of switches in the network are the so-called "don't care" switches. In such switches (e.g., switch 0 for a packet traveling to switch 6 in Figure 1(a)), all outgoing links lie on the shortest paths to the given destination. It means that a packet being routed to such a destination by a switch is never deflected.
- Mesh networks offer multiple paths between sources and destinations that increase the reliability of the network.

Datagram networks, in general, have one more advantage over connection-oriented networks. In the latter, a virtual connection is set between a pair of switches for the duration of the entire session. If some switches, used to relay traffic belonging to the given session get congested, more packets will be dropped, and the performance will degrade even if the neighbouring switches are idle. This problem does not exist in connection-less networks. If a datagram encounters congested switches on its path to the destination, it simply chooses another, less loaded route. This natural omission of "hot spots" adds even more flexibility in handling congestion in connection-less networks.

Disadvantages of deflection networks come from the fact that every packet is routed independently. It is thus impossible to avoid multiple paths, and misordering packets. For the same reason, it is impossible to determine the maximal length of the path, although there exist routing algorithms that provide a bound on the number of hops. In particular, the algorithm presented in [DG96] assures that the number of hops a packet can travel in a mesh, 2-connected network will never exceed $N-2$ where N is the number of switches in the network.

It was also proven in [CL91] that a randomized, distance routing algorithm guarantees the delivery of every packet within a finite network. This proof is based on the fact that a randomized routing algorithm ensures that the probability of deflection is less than one. It implies that all packets have a non-zero probability of traveling towards their destinations. Also, there is a maximum distance that the packet travels away from its destination. These facts, together with the assumption that a packet is removed at the destination, guarantee that the randomized, distance-based routing algorithm does not cause livelocks.

Since the resources of the network are not allocated in advance, it is hard to provide a quality of service. It may happen that due to a heavy load, new packets will not be allowed to enter the network. This may not have a serious impact on data transmission but may impair delay- and jitter-sensitive traffic.

Traditional approaches to multicast do not work in deflection networks. This is because such methods usually rely on replications at some switches. In the deflection network, packets belonging to the same session may follow different paths through different switches so it is rather hard to predict from the beginning where packets can be replicated. Moreover, replication is not possible when all outgoing links are busy (there are no buffers to store the replicas).

1.4 Outline of the Dissertation

We have identified the primary features of high-speed networks. We have introduced deflection networks and pointed out that, if not for some obstacles, they could be suitable for high transmission rates.

In the next chapters, we propose some ways to overcome these obstacles, showing that connection-less networks, exemplified by deflection networks, can satisfy the requirements posed by high-speed environments.

First, in Chapter 2, we show that jitter in deflection networks tends to be modest and that it remains similar regardless of the background traffic. This suggests that reassembly (playout) buffers of a reasonable size can smooth out the jitter assuring acceptable packet loss. This also suggests that running real-time applications, like videophone in deflection networks, is conceivable.

In Chapter 3, we present several simple schemes that solve the problem of multicasting in deflection networks. As we have already mentioned, some applications may require this capability and a high-speed network should be able to provide it.

Using our observations from Chapter 2 and the multicast algorithms proposed in Chapter 3, Chapter 4 discusses the performance of deflection networks in three "challenging", high-speed applications: videophone, videoconference, and transmission of video movies.

Having shown that with a limited network load and a sufficient size of reassembly buffers we can run video applications in deflection networks, in Chapter 5, we will make first attempts to provide a quality of service. We will look at a few statistical means of throttling the data traffic and giving a better service to the synchronous traffic.

A more complex but efficient way of providing quality of service is presented in Chapter 6. We shall see that by limiting the load and having a flexible adjustment of the playout buffer we are able to sustain the required long-term packet loss and end-to-end delay.

In Chapter 7 we analyze a more realistic, asynchronous version of deflection networks. We focus on the impact of the buffer space on their performance.

Finally, in Chapter 8, we compare the performance of deflection networks with the performance of store-and-forward networks. We will see that for a given amount of the buffer space used by the entire network, a deflection network offers much better performance than a store-and-forward network.

We will conclude this dissertation and list a few suggestions for future research in Chapter 9.

Chapter 2

Real-time Traffic in Deflection Networks

In this chapter (see also [OG98d]), we investigate experimentally the performance of deflection networks for real-time, jitter-sensitive traffic. Our goal is to show that despite the fact that packets often arrive misordered, jitter in a deflection network is reasonable, even when the traffic is relatively "unfriendly".

Some work on the reassembly buffer space in deflection networks was presented in [CM91]. In that paper, the authors studied the effect of a finite reassembly buffer on the performance of deflection routing under a specific non-uniform traffic model. We report the results of simulation experiments aimed at determining the buffer space requirements in deflection networks for reassembling ordered streams of packets at their destinations. We use the *jitter* (understood as the standard deviation of packet delay at the destination expressed in slots) as the measure of disorder introduced by deflection operating under different conditions. It turns out that even under malicious conditions, reasonable deflection networks tend to offer quite acceptable service to sessions that require packets to arrive at regular intervals and in the proper order.

2.1 Network Model

In addition to torus and triangle topologies shown in Figure 1.1, we consider a chordal ring (Figure 2.1). The ring, like the torus, is regular but its reachability is much poorer. The average shortest path length between a pair of switches is higher in a ring than in a torus with a comparable number of switches.

We first assume that the total delay involved in a single hop in the network is the same

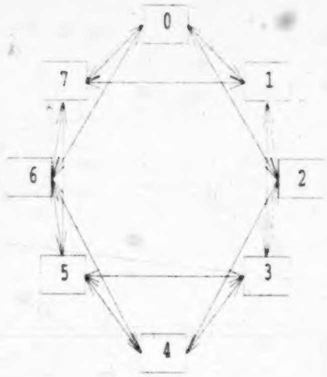


Figure 2.1: Ring topology, size 8, connectivity 4

for all links and equal to a single slot.[1] There are two selected switches: the source (S) and the destination (D). S sends its packets only to D; D receives packets only from S. We investigate the performance of the traffic session between the two selected switches under different background traffic scenarios in the remaining part of the network.

Each transmitting switch (that is, S and D) generates packets at some definite rate expressed in *1/slots*. For example, the fixed arrival rate of 1/3 means that a new packet is generated every three slots. Such a packet will be expedited in the first free slot arriving from the network. The selected source generates packets at a fixed rate. Our objective is to monitor the standard deviation of the interarrival time between consecutive packets observed at the selected destination. Transmission rules for sources other than S depend on the selection of the background traffic and will be explained later.

As soon as a packet sent by S arrives at D, its delay and jitter are computed. The delay is measured from the moment when the packet's predecessor was received, and the jitter is the standard deviation of the delay. These measures are well defined because in our model (and intentionally in all pure deflection networks) packets are never dropped at intermediate switches. Note, however, that packet delay (as defined above) can take negative values, which happens when a packet and its predecessor have been misordered. No performance measures are calculated for the background traffic.

To clarify the method of jitter computation, let us consider the following example:

```
time   2 4 8 12 14
number 1 3 2 4  5
```

---

[1] Results for longer links and for links with unequal lengths are presented in Section 2.2.5.

The first line shows the arrival times of packets, and the second line shows their sequential numbers. If d_{ij} denotes the interarrival delay between packets with sequential numbers i and j, then the delays in our example are as follows: $d_{12} = 6$, $d_{23} = -4$, $d_{34} = 8$, $d_{45} = 2$. Note that the interarrival delay between packets 2 and 3 is negative because packet 3 arrived before packet 2. The average interarrival delay is equal to $E(d_{ij}) = 3$. To find the jitter, i.e., standard deviation of interarrival delay, we have to compute the square root of variance $\sigma^2 = E(d_{ij}^2) - E(d_{ij})^2$, where $E(d_{ij}^2)$ is the moment of order 2 of the random variable d_{ij}. Thus, $\sigma^2 = 30 - 9 = 21$, and the jitter is $\sigma = 4.58$.

2.2 Results

In this section we present some of our simulation results obtained for the three reference networks. We consider large and small networks. A "large" network consists of $N = 256$ (torus and ring) or $N = 257$ (triangle) switches. Note that 257 is the closest approximation of 256 for which the triangle topology is complete. A small network is built of 1/4 of the number of switches constituting its large counterpart. Thus, it has $N = 64$ switches in the torus and ring cases, and $N = 65$ switches for the triangle. The connectivity of the torus and triangle (the number of link pairs per switch) is 2 and 4, whereas for the ring it is 4 and 8.[2]

We have carried out experiments for several different locations of the selected source and destination. For the results presented in this chapter, the two switches have been chosen as the two most distant switches in the network (thus we are looking at the worst case). For the torus and ring (the topologically symmetric networks), any pair of antipodal switches has this property and all such pairs are equivalent. For the triangle, in the "large" edition of the network, the sender is switch 17 (for connectivity $k = 2$) and 41 (for $k = 4$), and the recipient is switch 250 ($k = 2$) and 166 ($k = 4$).

2.2.1 Poisson Background Traffic

In this scenario, the background traffic during a simulation is uniform, with all sources and destinations (except S and D that do not contribute to the background traffic) being equally probable. Every switch other than S and D generates packets according to the Poisson distribution with a given mean. This mean is a load of the network (and the horizontal axis in the performance graphs) which determines how many new packets appear in the network within one slot time unit. Generated packets are then inserted to the transmitter queues of the randomly selected switches. S and D transmit packets to each other at the fixed rate as was explained above.

---

[2] Connectivity 2 is not very attractive for a ring as such a network looks like a poor cousin of FDDI deflecting packets in circles.

Packets that cannot be expedited immediately (because of the unavailability of free slots) are stored in queues at their source switches.

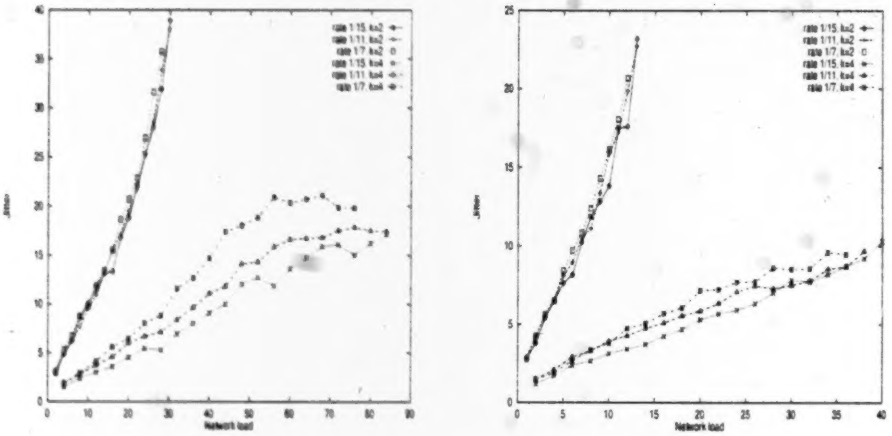
Figures 2.2 and 2.3 (note different scales) show the jitter vs background load for different transmission rates of the selected source S and different connectivities (k). As expected, the jitter increases with the increasing background load. Also, the transmission rate of the source has a visible impact on the jitter, especially in the torus with higher connectivity. This is somewhat surprising at first sight, because one would expect that higher-connectivity/reachability networks should offer not only lower jitter, but also better stability. It seems that sometimes the abundance of alternative routes with slightly varying shortest-path costs is not an advantageous property from the viewpoint of containing the jitter.

Nonetheless, although low connectivity/reachability networks sometimes offer better stability, they consistently incur significantly higher jitter than their better connected counterparts, even if the background traffic is not bursty and relatively smooth. Note that the connectivity-2 torus network is in fact the standard architecture of MSN.

As the network becomes larger, the jitter seems to grow proportionally to the increasing load (a larger network can carry more load). This phenomenon results from two factors: the increased distance between the two peers involved in the monitored session, and the increased opportunities for deflected packets to wander through remote regions of the network. Experiments indicate that the jitter depends primarily on the distance and increases only slightly with the network size, if the distance and (normalized) load in the neighbourhood remain fixed.

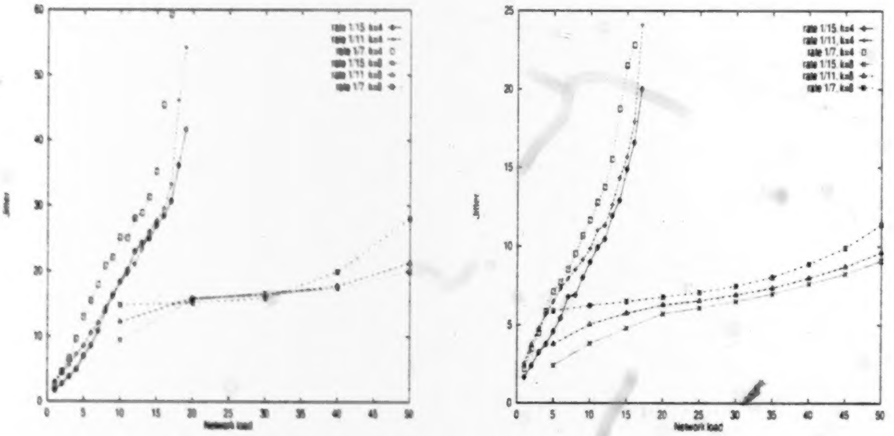
2.2.2 Correlated Poisson Background Traffic

In this scenario, the uniformity of the Poisson traffic model is disrupted by two additional parameters l (session length) and P (correlation level). Every l slots, a set of source-destination pairs is selected at random, with the number of pairs $n = P \times \frac{N}{2}$. For the next l slots, each of the selected senders will be generating uniform Poisson traffic addressed to its one dedicated destination. The remaining background switches (i.e., the ones that have not been selected), will carry on as before, generating Poisson traffic to randomly chosen destinations from outside of the selected set.



(a) Torus, N=256

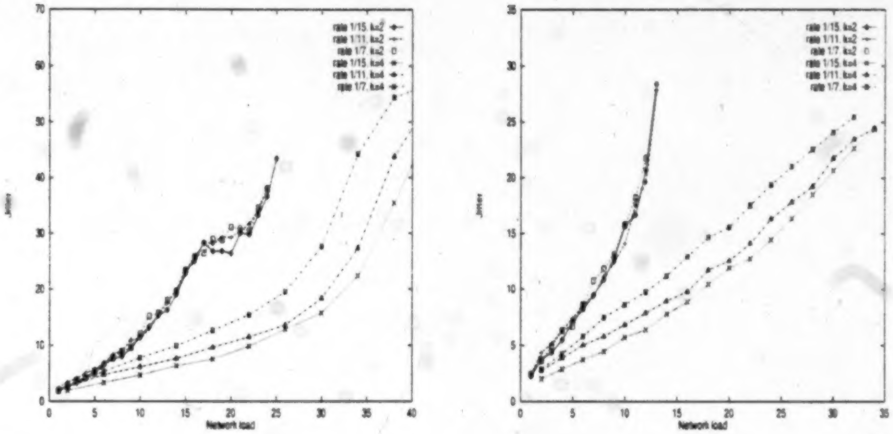
(b) Torus, N=64



(c) Ring, N=256

(d) Ring, N=64

Figure 2.2: Uniform Poisson background traffic in torus and ring networks



(a) Triangle, N=257

(b) Triangle, N=65

Figure 2.3: Uniform Poisson background traffic in triangle network

The intention of this traffic model is to represent scenarios in which there is a sustained traffic of non-trivial duration between pairs of switches engaged in some correlated sessions. The background load of the network can still be characterized by the global arrival rate of background packets.

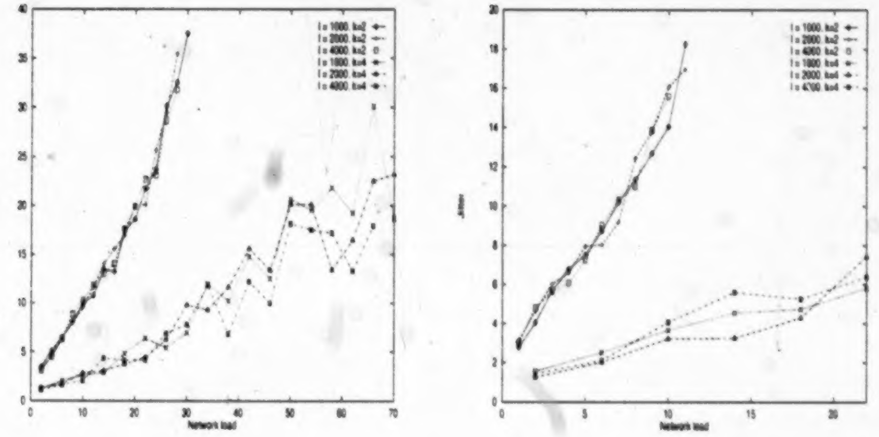
The results of these experiments are very close to the previous ones for practically all ranges of l and P (see Figures 2.4 and 2.5).

This indicates that deflection networks are not very sensitive to the changing patterns of loads, at least for as long as the global load in the neighbourhood remains at the same level. This property is advantageous for large networks, because their global loads tend to change slowly, although there may be local bursts of correlated activities.

2.2.3 Bursty Synchronous Background Traffic

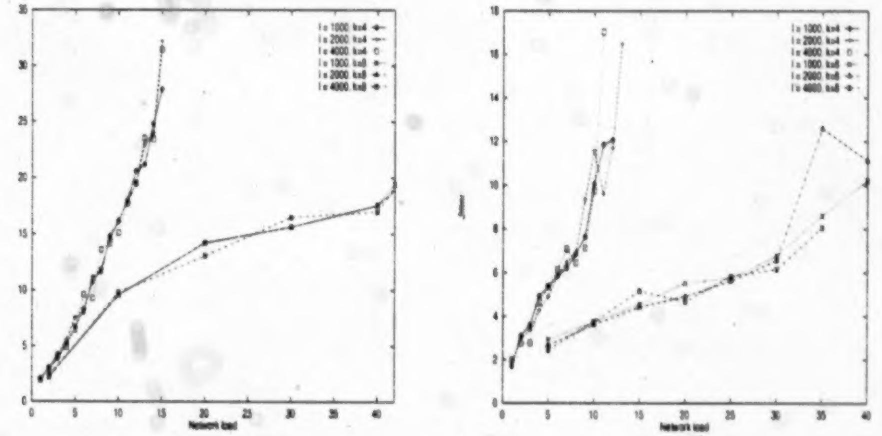
The intention of this scenario is to model a situation in which the background switches generate independent bursts of traffic transmitted at the highest possible rate. Intuitively, such a scenario should be malicious for the monitored synchronous session, because the interference is heavy and essentially non-deterministic.

Specifically, the traffic generator at every background switch operates as follows. It sleeps for an exponentially distributed random amount of time with mean t. After that time, it generates a burst of packets, its number determined by an exponentially distributed random number with mean n. All these packets are sent at the highest available rate. The



(a) Torus, N=256

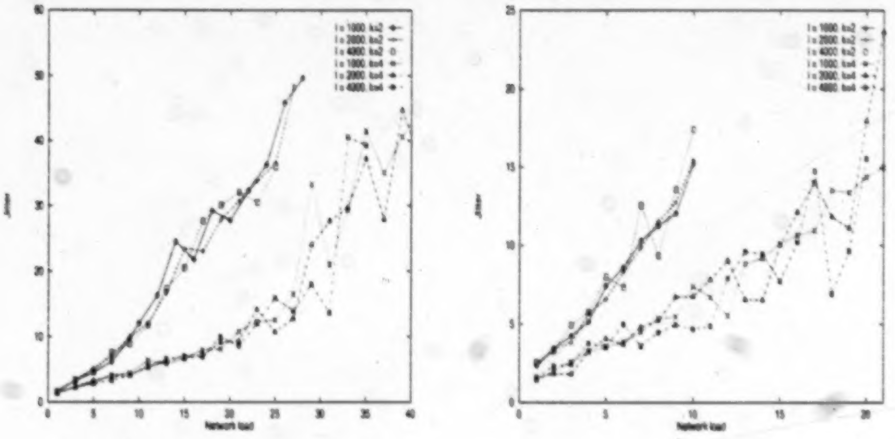
(b) Torus, N=64



(c) Ring N=256

(d) Ring N=64

Figure 2.4: Correlated Poisson background traffic in torus and ring networks of; $P = 0.4$, rate of S is $1/14$, $n = 51$ (in larger network) or $n = 12$ (in smaller network)



(a) Triangle N=257

(b) Triangle N=65

Figure 2.5: Correlated Poisson background traffic in triangle network; $P = 0.4$, rate of S is $1/14$, $n = 51$ (in larger network) or $n = 12$ (in smaller network)

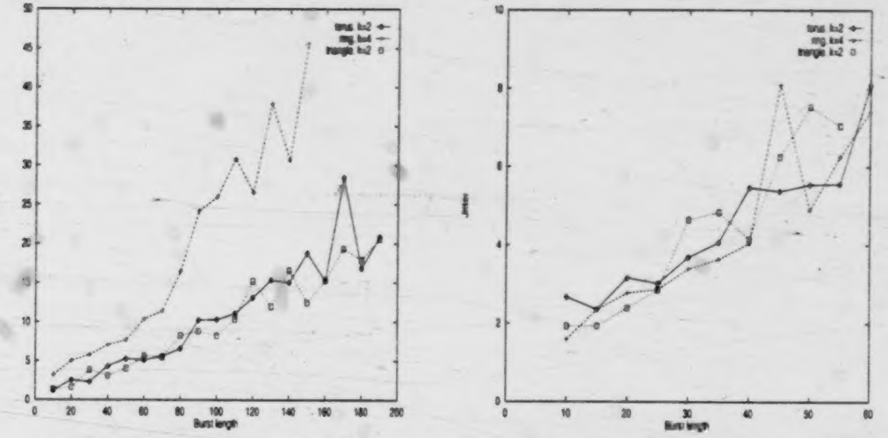
intensity of the background traffic is adjusted by modifying n, i.e., more intense traffic means longer bursts rather than a shorter interarrival time between bursts.

The results for this setup are shown in Figure 2.6. Notably, the observed jitter is not worse than for the other scenarios, except for the large ring, which seems to be most heavily penalized by the nondeterminism and burstiness of the background traffic. Note that large rings are not well suited for deflection because of the large deflection penalty, which increases rapidly with the increasing network size.

2.2.4 Other Traffic Patterns

We have investigated other background traffic scenarios, including long sustained synchronous sessions, self-similar bursty patterns, and bursty correlated patterns, as well as mixes of different traffic types. None of those experiments produced results worse than those mentioned in the preceding sections.

Generally, for the increasing network size, when the contribution of an individual session counts less in the total statistical mix, the observed jitter for the monitored synchronous session tends to remain low. Of course one can devise artificial sessions especially designed to interfere in the worst possible way with the monitored session, but one can do the same for store-and-forward networks, including ATM. Therefore, we were not interested in demonstrating that statistical fluctuations may produce arbitrarily large deviations in the



(a) Large networks

(b) Small networks

Figure 2.6: Bursty synchronous background traffic in networks of all three topologies and all sizes ($t = 20$)

observed hop count (which is obvious and trivial), but rather in determining how easy it is to stumble upon such configuration in a mix of background sessions with some statistical and reasonably unfriendly properties.

It turns out that it is not easy to produce really bad scenarios by accident, or even intentionally. In particular, bursty background traffic scenarios are generally much less unfriendly than one would expect. If the bursts are short, then the background traffic appears relatively smooth and, although the monitored synchronous session is disturbed, this disturbance is not long enough to worsen the jitter significantly. On the other hand, a long burst may disturb a few packets at the beginning, but later, while it persists, the network will learn to circumvent it and the disruption will cease to affect the jitter in a significant way.

2.2.5 Larger Propagation Delays

So far, we have assumed that propagation delay expressed in slots is low and equal to one slot. One can argue that in a high-speed network of a non-trivial size, the delay suffered by a packet on its single hop may be considerably longer. This may increase the jitter and buffer requirements at the destination.

Intuitively, if the delay and jitter are normalized to hops, their actual values can be obtained by multiplying the normalized values by the (average) channel length expressed as the propagation time. Let a *diameter* of a mesh network be the maximum length of the shortest path between a pair of switches. Figure 2.7 shows the jitter vs load for the uniform Poisson background scenario, with the source transmission rate 1/15, for the propagation delays between neighbouring switches equal to $l = 1, 5, 10, 15$ slots. The network is the connectivity-2 torus with 256 switches, and its diameter equals 17 hops. If we assume, e.g., as in [CM91], that the link capacity is 150 Mb/s, the slot length is 53 bytes, and the speed of signals in the medium is $0.69c$, then there are approximately 1.7 slots in transit on each kilometer of the link. It means that the propagation delay of 1 slot corresponds to a LAN of diameter 10 km.

The propagation delays of 5, 10, and 15 slots correspond to MANs of diameters 50, 100, and 150 km, respectively.

Alternatively, we can increase the network capacity 5, 10, and 15 times achieving 750 Mb/s, 1.5 Gb/s, and 2.25 Gb/s, respectively. The network diameters expressed in kilometers will be then equal to 2, 1, and 0.7 kilometers, respectively.

As expected, the jitter increases with the increasing propagation length of a link (l). To understand why the increase illustrated in Figure 2.7 does not appear exactly proportional to the link length (at least for the shorter links), let us consider the following.

The delay (in slots) of a single packet traveling through the network can be expressed as

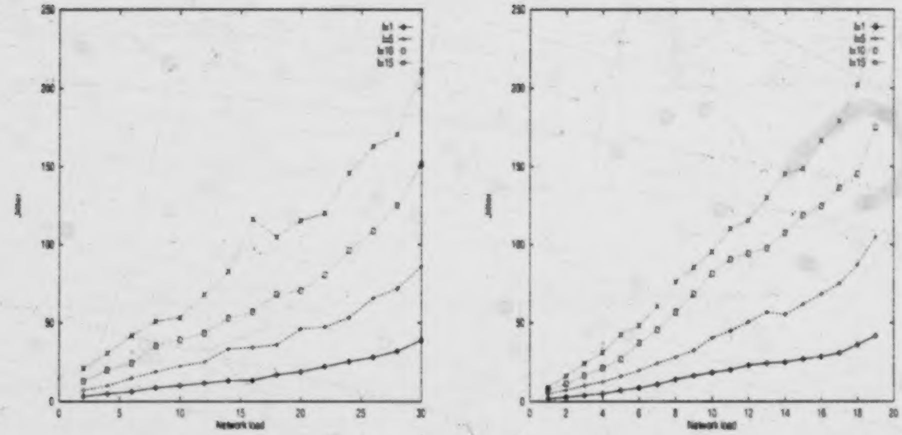
$$T = \overline{h} * (l + p) + \alpha * d * (l + p)$$

where l is the (average) propagation delay of a link, $\overline{h}$ is the average number of hops made by the packet on its way from the source to the destination, p is the processing time in the switch, d is the deflection index, i.e., the number of hops a single deflection adds to the packet's path, and α is the number of deflections accounting for the departure of the actual delay from the average.

In our experiments, $p = 1$ and $d = 4$ (deflection index in mesh networks with connectivity 2 is 4 regardless of the network size). α is a random variable with mean 0, bounded on the left by the negative average number of deflections and unbounded on the right.

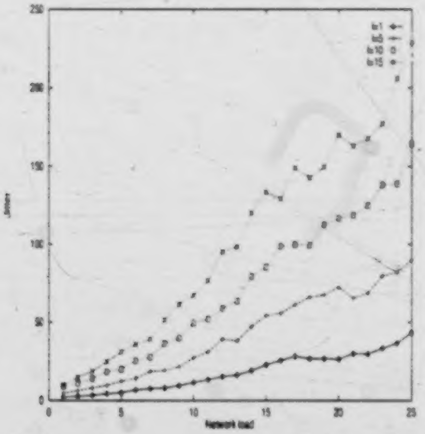
The nonzero jitter can now be viewed as a consequence of the fact that α is a random variable rather than constant. Thus, the variance σ^2 of the interarrival delays of M packets is

$$\sigma^2 = \frac{1}{M} * \sum_{i=1}^{M} (\alpha_i * d * (l + p))^2$$



(a) Torus

(b) Ring



(c) Triangle

Figure 2.7: Uniform Poisson background traffic for different propagation delays in large networks of different topologies (rate of S is 1/15)

and its standard deviation (jitter)

$$\sigma = d * (l + p) * \sqrt{\frac{\sum_{i=1}^{M} \alpha_i^2}{M}} \tag{2.1}$$

If the propagation delay l is increased by the factor of C, we get

$$\sigma_C = d * (C * l + p) * \sqrt{\frac{\sum_{i=1}^{M} \alpha_i^2}{M}} \tag{2.2}$$

Thus, the ratio of jitter increase when the propagation delays increase in ratio C is

$$F(l) = \frac{\sigma_C}{\sigma} = \frac{C * l + p}{l + p} < C$$

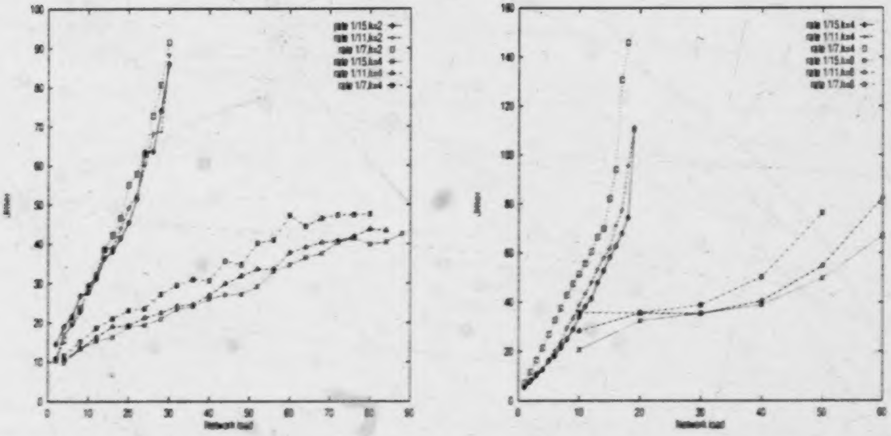
Note that we assume that $\sum_{i=1}^{M} \alpha_i^2$ in equations 2.1 and 2.2 is the same, which in general does not have to be true. Increasing propagation delays changes the environment of the network. A packet suffering some number of deflections in a network with $l = 1$ may be deflected a completely different number of times in a network with $l = 5$. In fact, the goal of this part of our experiments (which by the observation made at the beginning of this section might seem redundant) was to confirm that the distribution of α does not depend on the propagation length of the channels.

So far, we have considered networks in which all links have identical lengths. Let us now change this assumption, allowing the links to have different lengths. More specifically, every link between a pair of neighbouring switches has the length uniformly distributed from 1 to 10 slots. Results for the large network, Poisson background traffic, and all three topologies are presented in Figure 2.8.

If we compare the plots from Figure 2.8 with the plots from figures 2.2a, 2.2c, and 2.3a, we will notice that the relations between the plots for different connectivities and transmission rates of the source are very similar. The fact that links have non-equal lengths does not seem to affect the relation between the jitter and the network load.

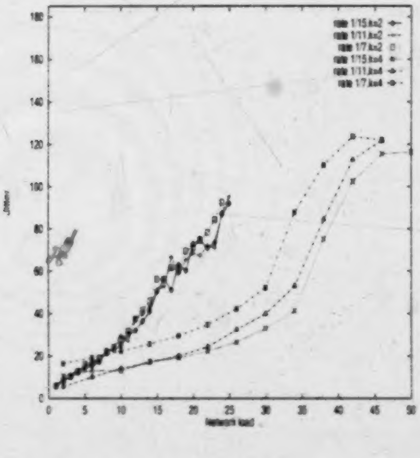
Now, let us compare the plots from Figure 2.8 with plots from Figure 2.7. We can see that the results obtained from the network in which links have lengths evenly distributed from 1 to 10 are in fact very similar to the results obtained from the network in which all links are identical and equal to 5 slots.

This all indicates that as far as jitter is concerned, the performance of deflection networks does not change regardless of whether the links between the neighbouring switches have identical or unequal lengths.



(a) Torus

(b) Ring



(c) Triangle

Figure 2.8: Uniform Poisson background traffic in the large networks with links of random lengths

2.2.6 Reassembly Buffers

Typically, the quality of service requirements of a synchronous session are specified in terms of packet loss rate rather than jitter. From the viewpoint of measurement, the jitter is a more convenient parameter to investigate, because under normal conditions not too many packets are lost, and it may take a long time to collect enough statistical samples to make a meaningful statement. It turns out, however, that the jitter can give us a fairly good idea of the packet loss rate for a given size of the reassembly buffer.

Our reception model operates as follows. Every packet arriving at the destination is first inserted in the reassembly buffer of size R. As soon as the number of packets in the buffer reaches $L = F \times R$ ($0 < F < 1$), the destination starts to "receive" packets (i.e., extract them from the buffer) at the rate equal to the transmission rate. The destination tries to receive the packets in the same order in which they have been sent. If a packet is not available when its turn comes, it is marked as lost. When that packet arrives later from the network it will be discarded and not stored in the buffer. Similarly, a packet arriving while the reassembly buffer is full, is dropped immediately. The role of L is to provide an initial backlog of packets to be received—to compensate for the variability and unpredictability of network delays.

At first sight, it would seem that F should be set to 0.5, with one half of the buffer space used to compensate for the unpredictability of delays in the "late" direction, and the other half used to buffer packets arriving early. This would be the case if the distribution of α were symmetric, which it clearly is not. Our experiments consistently indicate that the best value of F (for all deflection networks and traffic patterns) is about 0.85. This means that it is more important to account for those packets that have been delayed in the network than for those that may find the reassembly buffer full upon their arrival.

Numerous simulation results indicate that for R equal twice the observed jitter, practically no packets are ever lost (the loss rate is statistically unmeasurable and below any sensible QoS requirements for synchronous traffic). The loss rate of 1% typically occurs for R equal to the jitter and drops very fast as more buffer space is added. For a given percentage of packet loss, the buffer size is a linear function of the jitter.

2.3 Summary

We have presented some experimental results hinting at the expected jitter (and consequently buffer space requirements) in deflection networks used to carry traffic with timing constraints. We find these results optimistic. For example, consider a "large" torus network with the average propagation distance between a pair of neighbouring switches equal to 10 slots. Assuming the transmission rate of 150 Mb/s, this translates into the network diameter of 100 km. With the reassembly buffer size of 200, the network can cater to isochronous session, offering a very low packet loss rate. This buffer space is rather small (considering that the packet size of 53 bytes corresponds to the ATM cell) and allocated on a per session basis exclusively at the destination. Also, it has been arrived at under the assumption that the isochronous traffic receives no special treatment anywhere in the network.

Since the jitter is not large and the reassembly buffer of a reasonable size can smooth it out, we may expect that the performance of a deflection network in high-speed video applications will also be satisfactory. We investigate this issue in Chapter 4.

Chapter 3

Multicast in Deflection Networks

Multicasting in deflection networks is more difficult than in store-and-forward networks, because of the difficulties in sending multiple copies of the same packet on several output ports. On the other hand, deflected packets stray from their optimal paths and may visit the multicast recipients "accidentally," not necessary in the order envisioned by the sender.

As we have mentioned in Chapter 1, one important feature of a contemporary network is a multicast capability used in many applications, like video teleconferencing, video distribution, distributed computing and control. Unfortunately, traditional methods of multicasting do not work in deflection networks. The fact that packets cannot be buffered at the switches and the unpredictability of paths traveled by those packets make it impossible to use multicast algorithms applied in store-and-forward networks. For example, distance-vector multicast routing [FF62, DC90], or link-state multicast routing [MRR80, DC90], cannot be easily used in deflection networks because they are based on replication. If copies of a replicated packet cannot be sent over the desired outgoing links immediately, they are simply stored in buffers [Lie95]. This is difficult to do and unnatural in a deflection network, even if it uses some intermediate buffer space, because such space is usually limited [Max87]. Providing extra buffers just to accommodate multicasting would contradict the spirit of deflection; besides, this approach would produce rather unpredictable results depending on the background load.

Methods based on reverse path forwarding [DC90] cannot be directly used in deflection networks, not only because they rely on replication. Such methods assume that packets from a given source arrive at the given switch on the same input link. This assumption is used to discard multicast duplicates recognized as those packets that arrive on links different than the one appropriate for the given source. Of course, this simple trick cannot be used in a deflection network. Even in the complete absence of contention, a packet is free to choose among several paths with the same length.

It is thus clear that deflection networks require a different approach to the multicast problem than store-and-forward networks. In this chapter (see also [OG98b]), we present a number of simple multicast schemes that can be used with deflection. With our schemes, packet replication is unnecessary, although, as we shell see, it may improve their performance. Our multicast schemes also avoid the problem of duplicate recognition.

3.1 Network Model

In our simulation experiments, we have considered several network topologies, including torus and triangle topologies (see Figure 1.1). The results (at least in relative terms) were highly consistent across the different setups; therefore, we only present in this chapter the results for the torus network.

In our model, we assume that slots are never buffered at a switch, except for the alignment and routing, and that the total delay involved in a single hop in the network is the same for all links and equal to a single slot. By expressing all delays in hops, we normalize the results to the (average) length of the path in hops. Our experiments with networks using different (also non-identical) distances between neighbouring switches have confirmed the validity of this generalization.

There is a distinguished source $S = 0$ that generates multicast packets (also called *M-packets*). In a regular network, exemplified by the torus topology, the selection of that source is immaterial. An M-packet is addressed to some number D of multicast destinations. We assume that the list of those destinations is stored in the packet's header.[1]

For the sake of measurement, the source will send a new M-packet only if the previous one was received by *all D* destinations. This is unrealistic in a real network because a source cannot know exactly when a multicast packet has been received by its last destination. However, in this way we can investigate the behaviour of individual M-packets without making them compete among themselves. It also provides the same environment for different multicast schemes. In the next chapter, we will see how some of our multicast algorithms perform in a realistic, videoconference environment.

Our primary performance measure is the average delay suffered by a multicast packet on its way to the *last* recipient. This delay is expressed in hops, i.e., normalized to the (average) link length in the network. Note that this time, we do not measure jitter which is useful only in the context of stream-oriented, real-time applications. In Section 3.4.1 we also show the average length of the path from the source to every multicast recipient.

---

[1] In Section 3.3 we discuss methods of encoding this list efficiently.

First, we measure the delay versus the number D of multicast destinations in an empty network, i.e., in which the transmission of M-packets is the only activity. Then, we set D to some value and introduce a Poisson background traffic to the network. Under this scenario, every switch other than S generates packets addressed to a single destination randomly selected at the beginning of a simulation. Numerically, this background load is expressed as the number of new packets generated in the network during one slot time. Generated packets are stored in a queue at the source switch, and they are extracted from the queue at every opportunity, i.e., whenever an empty slot happens to be passing through the switch. No performance measures are collected for the background traffic—we only observe how this traffic affects the performance of the proposed multicast schemes.

3.2 Basic Multicast Schemes

Let us start from the following generic multicast strategy. The packet is transmitted by S—the distinguished multicast source—to the switch whose number is at the head of the list of destinations. This first destination is dubbed the *bounce destination*. Every intermediate switch on the path from S to the bounce destination checks if it is listed in the packet's header. If so, the switch receives the packet without removing it from the network, i.e., clears its entry on the list of destinations stored in the header. If the list is still nonempty, the packet is relayed on the outgoing link, according to the routing scheme. If the list is empty, the packet is removed from the network.

When the packet arrives at the bounce destination and the list of recipients is still nonempty, a new bounce destination is selected and the packet continues its trip. Note that during this process, the bounce destination may change several times (at most $D-1$ times).

The above generic strategy will be modified by changing the ways of selecting subsequent bounce destinations. Also, we will occasionally allow the packet to be replicated at a switch. Some of the possible variations are listed below.

B_Random

This is exactly the generic scheme presented above, in which the bounce destination is selected at random from the current list of recipients.

B_Shortest

In this scheme, the bounce destinations are ordered from the closest to the source to the most distant from the source.

B_Furthest

This approach is opposite to **B_Shortest**. The bounce destinations are ordered from the furthest from the source to the nearest to the source. The idea is to let the packet visit a few recipients "accidentally" before it gets to the first bounce destination.

B_Bounce

This scheme is similar to B_Shortest, except that the next bounce destination is chosen as the one being the closest to the current bounce destination. The first bounce destination is chosen as the one being the closest to the source.

B_All

This scheme resembles B_Bounce (and is identical to B_Bounce if there are no deflections). Any recipient (including "accidental recipients") is promoted to the status of a bounce destination. The next bounce destination is determined as the one being the closest to the current recipient.

3.3 Implementation

The above strategies assume no packet replication and, at least at first sight, can be viewed as having been presented in the increasing order of their complexity. Clearly, B_Random is the simplest strategy, as it requires practically no processing at the source or at the intermediate recipients.

To implement any of the basic strategies, we have to augment the packet header by two fields:

- multicast session identifier
- destination set

We assume that the header also includes two standard fields needed by regular traffic, i.e., the source and destination address.

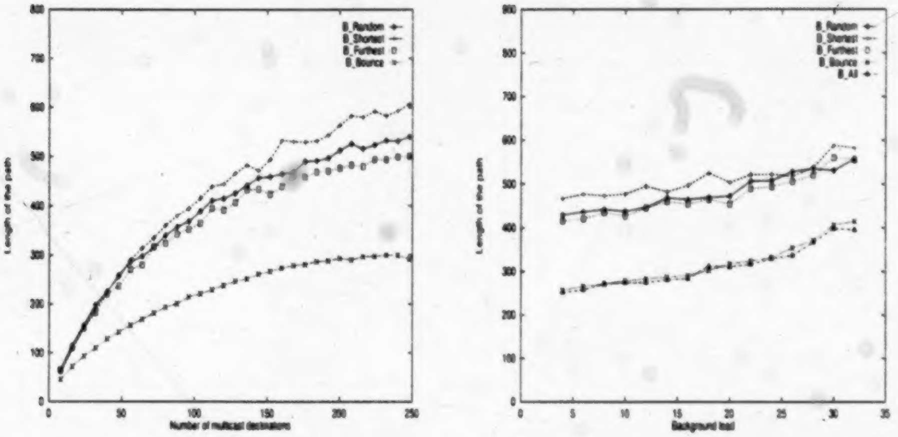
Upon the setup of a multicast session, the source selects a session identifier, which is supposed to be unique in the network. One part of that identifier can be the source Id, another part can be a source-selected number telling apart different multicast sessions carried out by this source.

Then the source notifies the recipients of the multicast session. Each recipient is assigned a multicast Id between 0 and D, where D is the total size of the recipient pool. This way, the range of the recipient Id is confined to the size of the population of recipients (rather than the number of stations in the network), which makes it easier and more economical to represent recipient sets as bit patterns.

We assume that a set of recipients is represented in a fixed-width binary field (e.g., 32, 64, 128 bits) with every bit position corresponding to one recipient. This solution imposes a limit on D. In Section 3.7 we discuss methods of overcoming this restriction without inflating the size of the destination set beyond a reasonable value. In asynchronous deflection networks, where packets can be of variable length, the size of the destination set may vary, depending on the actual population of the recipient pool.

The destination address field of an M-packet contains the address of the next bounce destination. This address is first determined by the source and later at every subsequent bounce destination. For the strategies B_Bounce and B_All, this operation is very natural. For the first two strategies, the current bounce destination determines the next bounce destination based on the source address extracted from the packet header. If the network is regular (or almost regular)—see [Max87]—this involves simple calculations on the row/column coordinates of the respective stations. With the assistance of a dedicated hardware, these calculations can be performed in parallel for all destinations remaining in the set. This is yet another reason for limiting the maximum size of the destination set to a "reasonable" number.

In this context, schemes B_Shortest and B_Furthest appear in fact more complex than the last two schemes. Our results (Section 3.4) indicate that B_Bounce and B_All are significantly better than the other proposed solutions; therefore, one can view B_Shortest and B_Furthest as only providing two more reference points for evaluation. In fact, B_Random is the only sensible competitor. Although it generally performs much worse than B_Bounce and B_All, it requires considerably less processing at the bounce destinations.



(a) Delay, D changes, k=2

(b) Delay, D constant, k=2

Figure 3.1: Basic schemes (no replication), connectivity 2

3.4 Results

In this section we present some of our simulation results obtained for a synchronous deflection network built on the torus topology. First, we observe the performance of basic multicast schemes presented in Section 3.2. Then we incorporate a simple replication technique and compare the performance of the schemes with and without their application.

The network has $N = 256$ switches, and its connectivity (the number of link pairs per switch) is $k = 2$ or $k = 4$.

3.4.1 Basic Schemes

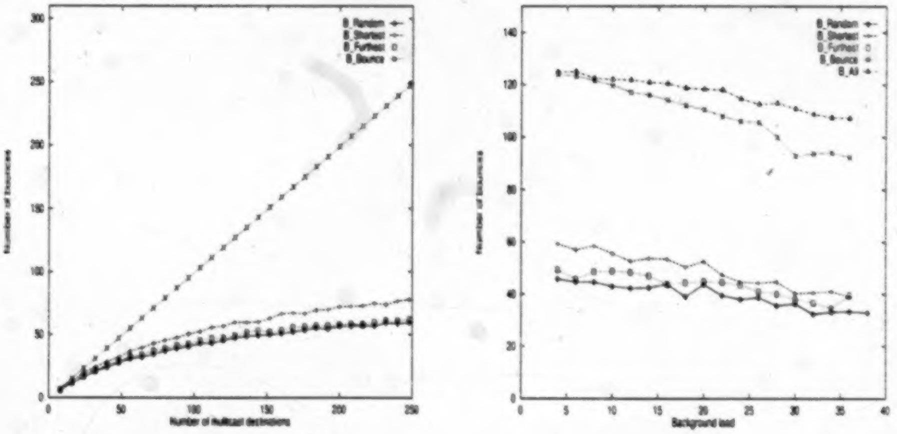
Connectivity-2 Networks

Figure 3.1 shows the delay vs the number D of multicast destinations when the background load is 0 (3.1a), and when the background load changes while D remains the same (3.1b). First, let us analyze the results shown in Figure 3.1a.

As expected, B_Bounce is much better than other schemes. Destinations in the packet's header are reordered from the nearest to the furthest at every bounce switch, with respect to that switch. If there is no contention, this allows a packet to move from one destination to another over the shortest path from among all the strategies considered in this chapter.[2] It obviously assures that the delay achieved in this scheme is the lowest.

---

[2] Note that B_All is identical to B_Bounce if there are no deflections.



(a) D changes, k=2

(b) D=128, k=2

Figure 3.2: Number of bounces vs number of multicast destinations and vs background load. Basic schemes, connectivity 2

Note that the path traveled by a packet with B_Bounce is not the shortest possible. Finding such a path would involve solving an instance of the traveling salesman problem, which we consider too complex for our kind of application.

Differences between the first three, simpler schemes are rather small. They are caused mainly by what an M-packet can do on its way to the first bounce switch. For a larger number of multicast destinations, B_Furthest is surprisingly the best among the simpler schemes. The reason for this is that the packet tends to visit "accidentally" a few other destinations on the list before it reaches the bounce destination to which it was addressed.

Figure 3.2a showing the number of times a packet visits a bounce destination (i.e., the destination to which it was addressed) vs D, seems to support this claim. Indeed, M-packet destinations ordered according to B_Furthest experience fewer bounces from destinations to which they were addressed. It means that they "accidentally" visit some other destinations stored on the list in the M-packet's header.

When there is no background load, the plot corresponding to B_Bounce is a straight line because the number of bounces is always $D - 1$. In such a case, an M-packet is always forwarded to the destination located closest to the forwarding bounce switch. Thus, they cannot stray and make unplanned visits to other destinations.

It is interesting to notice that despite the fact that B_Furthest outperforms B_Random, the number of bounces in the latter scheme is slightly smaller. It seems that the path of an M-packet in B_Random is so chaotic that it takes a packet more time to visit all destinations, and the fact that relatively many of them are reached "accidentally" is not able to outweigh that phenomenon.

Now, let us look at the results showing what happens when a background load is present in the network while the number of multicast destinations remains fixed at $D = 128$ (Figures 3.1b and 3.2b). These figures also present B_All, in which destinations are reordered in *all* intermediate destinations, not only in the bounce ones.

For low background loads, the differences between the simplest schemes are similar to what we have seen already. Then, when the load gets heavier, the differences tend to dissipate. Under such conditions, the number of deflections is so high that it is rather irrelevant where an M-packet is sent first—to the nearest, furthest, or random destination. Figure 3.2b, showing the number of bounces vs background load when $D = 128$, illustrates that situation.

B_Bounce is still better than the three simple schemes. However, for heavy loads it is slightly worse than B_All. The reason is clear. When the load is low, M-packets rarely stray from their shortest paths between the destinations, and both schemes are almost identical. When the number of deflections increases, M-packets are sometimes deflected to other destinations on the list. With B_Bounce, such a packet will try to return to the previous path leading to the bounce destination it was addressed to even if another destination from the list is just one hop away. With B_All, multicast destinations will be reordered and the packet will be forwarded to the nearest destination.

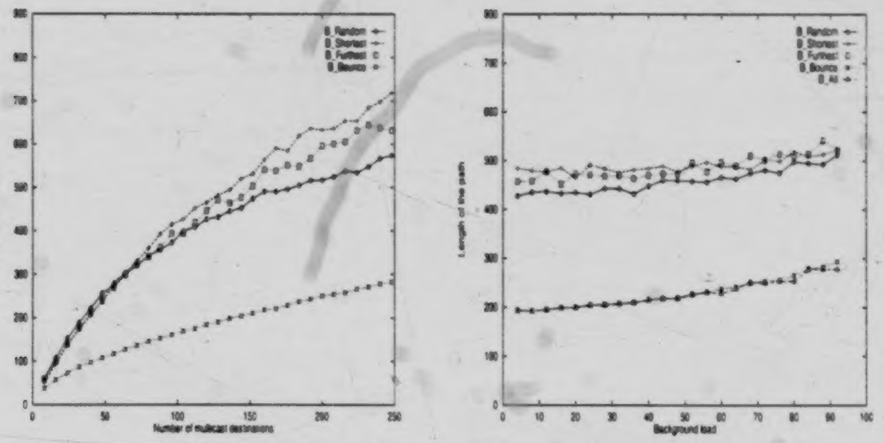
Among all the multicast algorithms presented above, B_Bounce significantly outperforms the others. B_All is slightly better in a heavily loaded network.

Connectivity 4

Figure 3.3 presents the results for the network with connectivity 4. It shows the delay vs the number D of multicast destinations when the background load is 0 (3.3a), and when the background load changes while D remains fixed (3.3b). Let us start from the results shown in Figure 3.3a.

Similar to what we have seen for the low connectivity network, B_Bounce offers the best performance. However, B_Furthest is no longer the best among the simple schemes. Its place in this category was surprisingly taken by B_Random. This can be explained as follows.

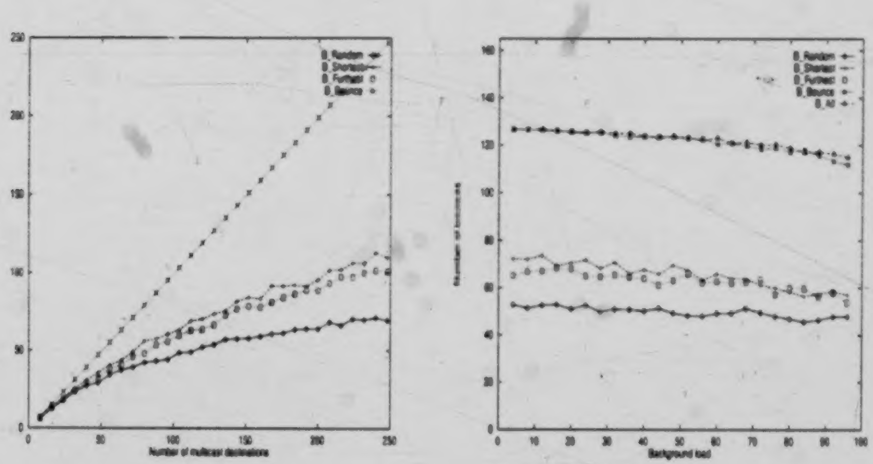
In a network with a high connectivity, the average path is shorter compared to the path



(a) Delay, D changes, k=4

(b) Delay, D constant, k=4

Figure 3.3: Delay vs number *D* of multicast destinations and vs background load. Basic schemes, connectivity 4



(a) D changes, k=4

(b) D=128, k=4

Figure 3.4: Number of bounces vs number of multicast destinations and vs background load. Basic schemes, connectivity 4

length in the same network with a lower connectivity. An M-packet can move faster from one destination to another, so the initial ordering of packets is less important. The chaotic path of an M packet that made B_Random worse for $k = 2$ is now beneficial. Because of this, the packet has more chances to be received by destinations other than the intended bounce one. Indeed, the number of bounces shown in Figure 3.4 is much smaller for the random scheme than for the other approaches. But why is it better than B_Furthest and B_Shortest?

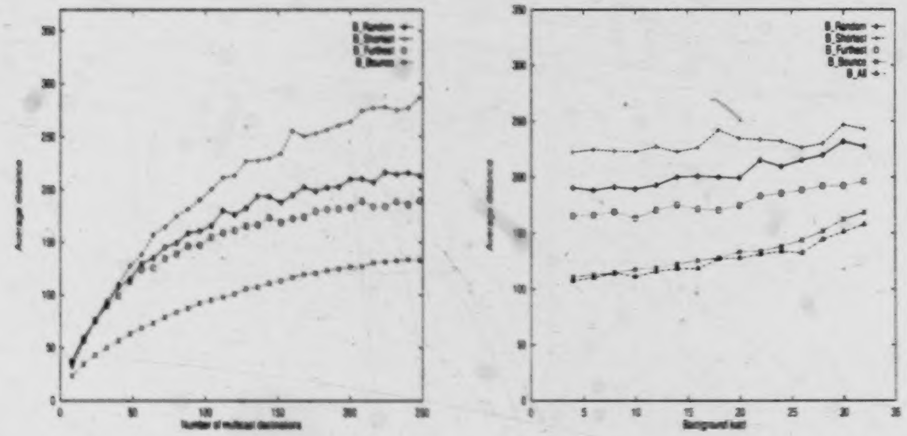
In B_Furthest, we rely on "accidental" visits of an M-packet to some of the multicast destinations. Here, such visits are less probable because the paths are shorter. The M-packet gets to the first destination in a fewer number of hops which decreases the probability of being received by some other destinations. Thus, the M-packet tends to be received by more distant destinations first, approaching gradually the source. The probability of visiting "accidentally" destinations other than the bounce one is further decreased in B_Shortest, which again makes this scheme poor.

It seems that if multicast destinations are ordered only in the source, the schemes that assure the small number of bounces, that is, the high number of receptions by destinations other than the bounce ones, perform better, particularly in a network with a high connectivity. In such a network, the average path that a packet must follow is relatively short, and even if the path is rather chaotic (like in the B_Random case), it does not decrease the performance. In a small connectivity network, where an average packet has to make more hops, the unordered path of a packet may overcome the benefits described above.

Among the multicast algorithms applied to the network with connectivity 4, B_Bounce significantly outperforms the simple schemes, but its performance is as good as B_All, even in a heavily loaded network. Since B_Bounce is less complex than B_All, B_Bounce appears to be better. If the low complexity of the scheme is important, B_Random seems to be a reasonable choice. In this scenario, it outperforms the remaining two simple (but more complex) schemes.

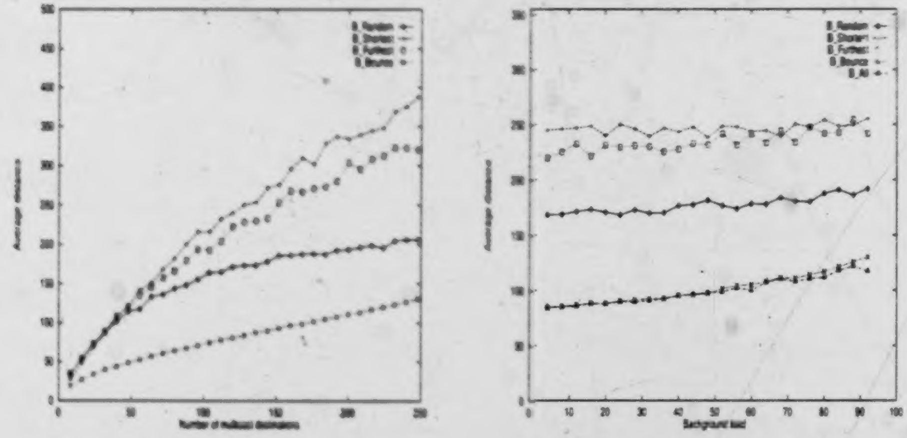
Average Length of the Path

The average path length versus the number of multicast destinations and background load is shown in Figure 3.5. The average distance for a given multicast scheme from the source to the destination reaches about half the total length of the path from the source to the last destination, shown in Figures 3.1 and 3.3. Relations between particular plots for different basic schemes are similar to the relations presented in those figures. In the following sections,



(a) Average path, D changes, k=2

(b) Average path, D constant, k=2



(c) Average path, D changes, k=4

((d) Average path, D constant, k=4

Figure 3.5: Average length of the path with basic schemes (no replication), connectivity 2 and 4

we will investigate only the delay understood as the length of the path from the source to the last destination.

3.4.2 Limited Replication Schemes

As we mentioned in Chapter 1, replication schemes in deflection networks are not very useful for the following two reasons:

1. a replication cannot be guaranteed, because the free slot may not be available
2. simple methods of rejecting packet replicates (used e.g., in reverse path forwarding) do not work, because packets from the same source may arrive on different input links

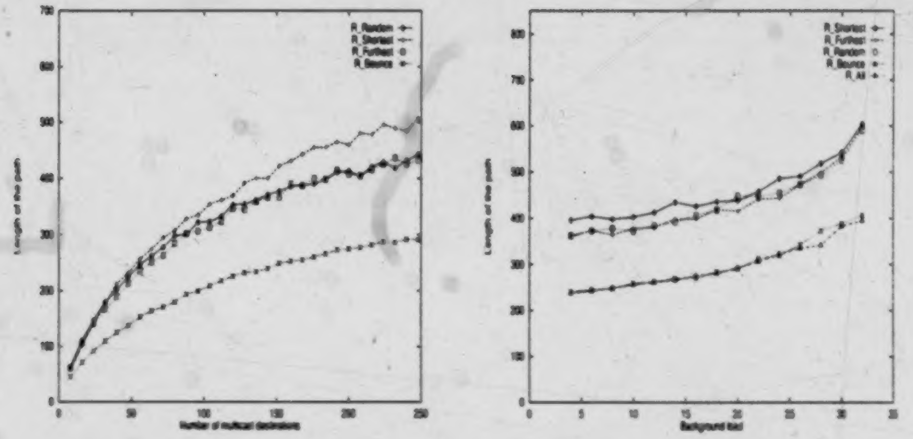
In this section, we incorporate a simple replication technique in our schemes. Namely, if it happens that the neighbour of the current switch appears on the list of destinations, and a free slot is available on the link connecting the current switch to that neighbour, the packet is replicated and addressed specifically to the neighbour. We know that such a packet will not be deflected and that it will reach the destination in one hop.

The schemes presented so far are now enhanced with the simple replication feature introduced above. It is expected that this scheme will decrease the time spent by the packet in network, thereby decreasing the delay, particularly under light loads. If the load is high, replications will occur less often because the outgoing links will be more likely to be busy.

The increase of the algorithm complexity is not high. As before, every switch must scan the list of destinations found in the M-packet's header. This time, it also has to find out if any destination on this list is among the switch's neighbours.

The performance of the replication schemes vs D, and vs background load when connectivity is 2 are presented in Figure 3.6. The first letter of each scheme's name is now 'R' to indicate that the scheme has been augmented by the replication feature. The relations between the replication schemes are similar to the relations between the basic schemes discussed in the previous section (Figure 3.1).

The difference between R_Random and R_Furthest is smaller than the difference between B_Random and B_Furthest. It seems that the chaotic path of an M-packet in a random scheme is even more beneficial than before. Note that in a replication scheme, a packet may be received by one of its destinations not only if it actually arrives at that destination, but also when it passes in its vicinity. With R_Random, a packet has more opportunities to pass in the neighbourhood of one of its destinations, and get replicated. However, this



(a) Delay, D changes, k=2

(b) Delay, D constant, k=2

Figure 3.6: Delay vs number D of multicast destinations and background load. Replication schemes, connectivity 2

phenomenon loses quickly its impact when the background load is significantly greater than zero (see Figure 3.6b). In such a case, R_Furthest performs better, although for higher loads, the differences dissipate again—as they did in the basic schemes.

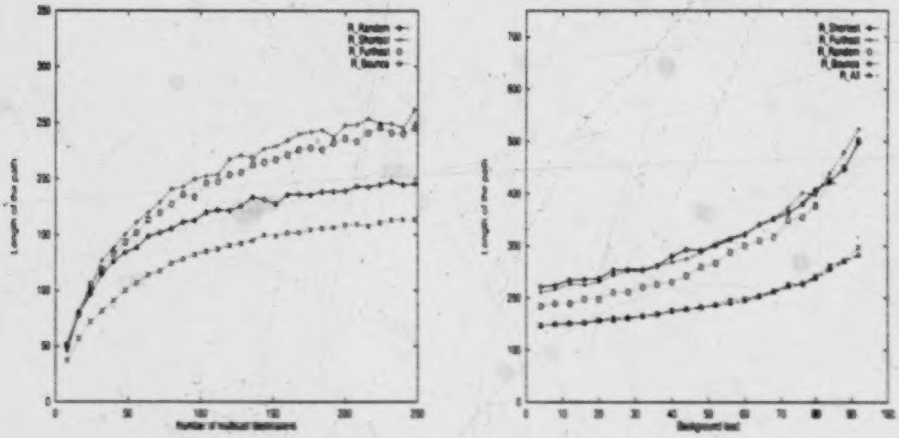
The performance of the replication schemes in the connectivity-4 network is presented in Figure 3.7. The relations between particular schemes are similar to what we have seen for connectivity 2. Notably, the difference between the performance of R_Random and R_Furthest becomes even more significant.

3.5 Comparison Between the Basic and Limited Replication Schemes

Figure 3.8 shows a comparison between the schemes presented in the preceding sections when the network connectivity is 2. The comparison is made between the best simple schemes (i.e., B_Random, R_Random), and the schemes that require packet ordering from some (or all) destinations (i.e., B_All, R_All). Note that in Figure 3.8a, B(R)_Bounce is shown rather than B(R)_All. When the background load is 0, these schemes are equivalent.

Among both the simple and more complex schemes, the ones based on replication are better. However, they involve some additional processing at every switch, and their benefits become questionable in a heavily loaded network (see Figure 3.8b).

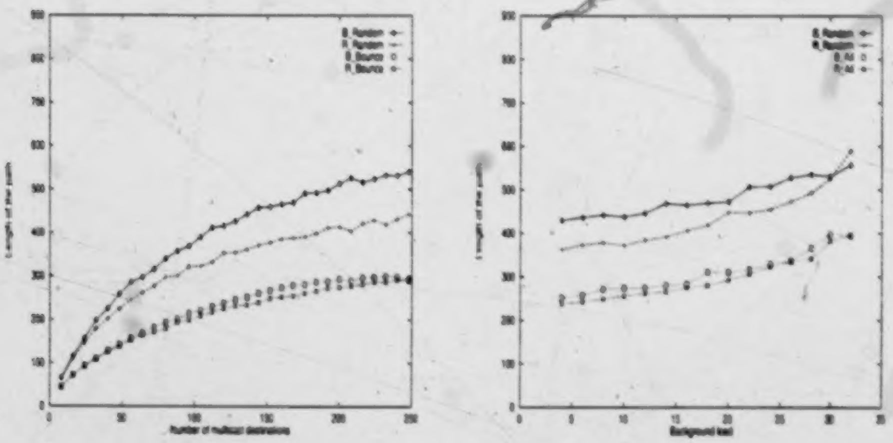
If the processing at switches is to be kept at a minimum, the basic schemes still perform



(a) Delay, D changes, k=4

(b) Delay, D constant, k=4

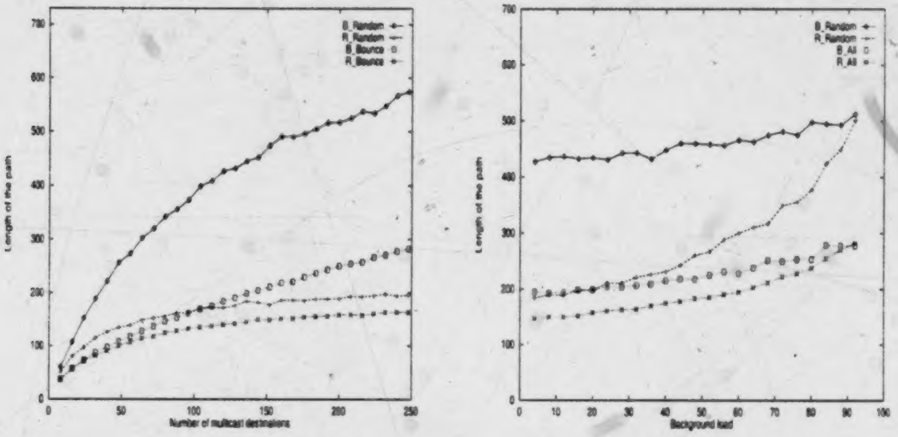
Figure 3.7: Delay vs number D of multicast destinations and background load. Replication schemes, connectivity 4



(a) Delay, D changes, k=2

(b) Delay, D constant, k=2

Figure 3.8: Delay vs number D of multicast destinations and vs background load. Comparison of the schemes, connectivity 2



(a) Delay, D changes, k=4

(b) Delay, D constant, k=4

Figure 3.9: Delay vs number D of multicast destinations and vs background load. Comparison between the schemes, connectivity 4

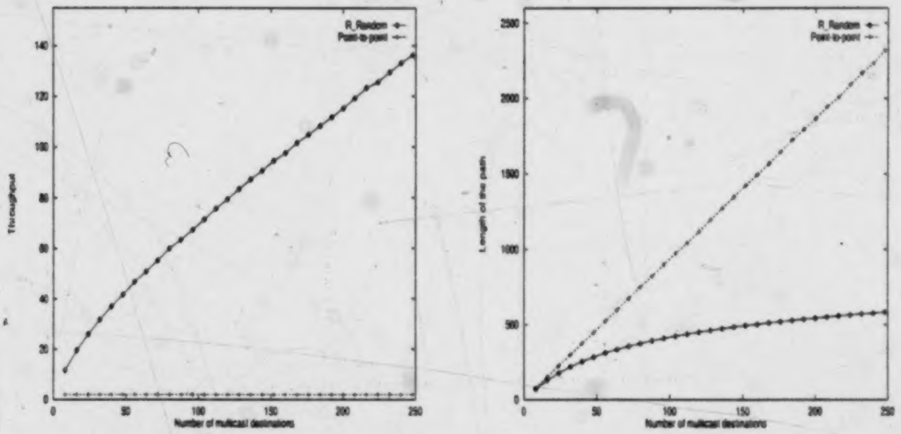
quite well. Their most complex member (B_All) also involves processing in some of the intermediate switches, but its complexity is still lower than that of the R schemes.

Figure 3.9 shows a comparison between the schemes presented in preceding sections when the network connectivity is 4. First, let us look at Figure 3.9a which shows the performance of the schemes with varying D.

Surprisingly, if the background load is 0 and D is big, R_Random is better than B_Bounce. When the connectivity is high (and the average path length is small) it is more beneficial to let the packet wander around in the network from destination to destination without any specific order. The path is short so it will not take the packet too much time to visit all destinations. At the same time, it will be more likely to visit some other destinations "accidentally," particularly if replication is in effect. Thus, we confront two mechanisms:

- potential length increase of the packet's path, but at the same time an increase of the probability of "accidental" arrivals due to the replication mechanism (R_Random);
- potential length decrease of the packet's path, but at the same time a decrease of the number of destinations that may be visited "accidentally" (B_Bounce).

The experiments indicate that the first mechanism gives better results when the background load is low and the network connectivity is high. Of course, the scheme using the advantages



(a) Throughput, k=2

(b) Delay, k=2

Figure 3.10: Throughput and delay vs number D of multicast destinations. Comparison between R_Random and point-to-point scheme

of both mechanisms (R_Bounce) performs best.

If we look at Figure 3.9b showing the performance of the schemes vs background load, we notice that the plots for the schemes relying on replication converge to those of the basic schemes when the background load increases. Clearly, with the increasing load, M-packets have fewer chances for being replicated which causes basic and random schemes to behave alike.

Let us now compare one of the above schemes, say R_Random, with the *point-to-point* scheme in which the source sends D separate packets to D destinations. In the absence of multicast techniques, this is what the source would have to do to deliver a given message to a number of receivers. We assume that the source has a continuous supply of packets sent every time the outgoing link is perceived free. That is, the source no longer waits until the message is received by all intended destinations but keeps on transmitting its packets. R_Random was modified in a similar fashion.

Figure 3.10 shows a comparison between the two schemes. The difference in throughput is particularly large. In the point-to-point scheme, sending a message to D destinations requires D packets—during the time needed to transmit them, the source using R_Random is able to send D different messages addressed to *all* multicast destinations. It clearly improves the throughput of R_Random compared with the point-to-point scheme.

The delay comparison between the two schemes is also beneficial for R_Random. Sending

D packets to D destination requires more hops than sending one packet to D destinations. The difference becomes more pronounced for higher D.

Similar observations were made when D was constant and the background load was changing, and when the network connectivity was 4.

3.6 Extended Replication Schemes

In section 3.4.2, we have presented schemes enhanced by a limited replication. In those schemes, an M-packet can be replicated at switch K if one of its destinations is one hop away from K and the appropriate outgoing link is free.

In this section, we will extend the limited replication (R_All scheme in particular). We will observe schemes that allow replicated fragments to be forwarded to destinations located further than one hop away from the switch at which a replication takes place.

3.6.1 Extended Replication

Let us define a *replication distance* R_d as a distance (in hops) between switch K and one of the destinations to which an M-packet is sent. The replication schemes presented so far have $R_d = 1$. Here, we present a model in which an M-packet may be replicated at K even if the distance between K and one of its destinations is greater than one hop, that is, in which a replication distance $R_d > 1$.

When an M-packet passes through switch K and some outgoing links are free, the switch examines distances to every destination d found in the packet's header through every free link. It attempts to find the destination whose distance over the free link is not greater than R_d and that is minimal among all distances from K to every d through the free link. If such a distance is found, the packet is replicated and its copy is relayed through the selected link. Note, that if the network's connectivity is greater than 2, the packet may be replicated into more fragments at the same switch.

In other words, destination d of a fragment that comes from a replicated packet, and the link through which it is transmitted are selected to minimize the distance between K and d.

For example, let us see how an M-packet addressed to multicast destinations: 1, 2, 17, 18, 32, and 33 will be routed at switch 0 (Figure 3.11). Assuming that both outgoing links are free, and replication distance $R_d = 2$, the packet will be divided into two fragments. One fragment addressed to destination 1 will be routed over the link A, while the other fragment will be routed to the remaining destinations over the link B. This is because destination

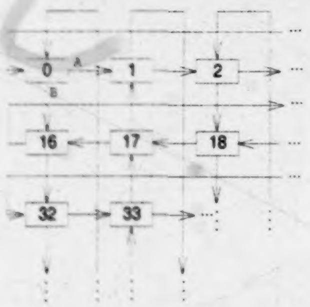


Figure 3.11: A fragment of a 16x16 torus network

1 is closest to switch 0 among all other destinations in the packet's header. Note that if a packet were addressed only to destinations 17, 18, and 33, it would not be replicated at switch 0 because the distance from 0 to these switches exceeds R_d.

We will call this algorithm extended (*R_Ext*). Note, that if $R_d = 1$, this scheme is equivalent to R_All.

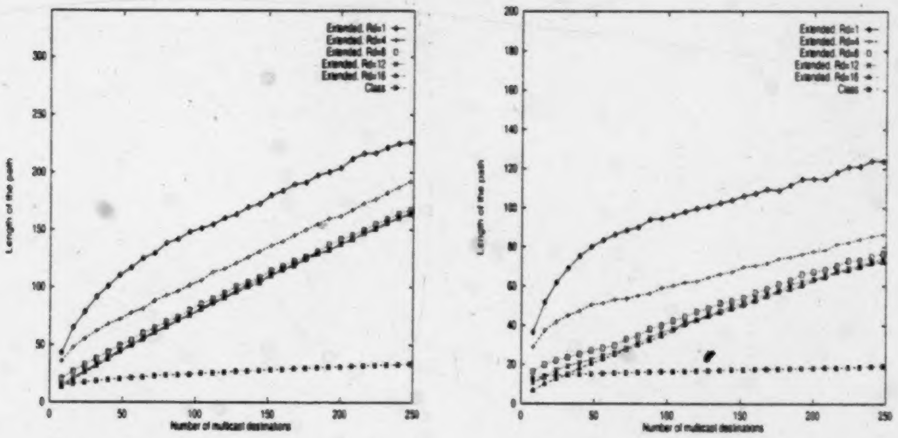
3.6.2 Class Replication

Yet another replication algorithm considered in our thesis is called *class replication* (*R_Class*). If an M-packet passes through switch K and $k \geq n > 0$ links are free (where k is the connectivity), it may be divided into up to $n - 1$ new M-packets such that every new packet contains addresses of destinations that are closest through the given link.

The switch selects an M-packet with the largest number of destinations, computes the distance from K to destination d through every free link. It then writes the address of d to the (possibly newly created) packet that will be routed through the link that gives the shortest path to d. The same process is repeated for every destination in the packet's header. In this way, destinations are divided into a number of classes determined by the number of free links and the addresses of destinations.

Note that it is possible that no new packet will be created. It happens if the distances from K to all destinations in the packet's header are minimal through the same link.

To better understand this scheme, let us again consider the fate of an M-packet routed at switch 0, addressed to switches 1, 2, 17, 18, 32, and 33 (Figure 3.11). Assuming that both outgoing links are free, the packet will be again divided into two fragments. This time however, the fragment routed over link A will be addressed to destinations 1, 2, 17, and 18, while the fragment routed over the link B will be addressed to 32 and 33. This is



(a) k=2

(b) k=4

Figure 3.12: Delay vs number D for different replication distances R_d in R_Ext, and in R_Class

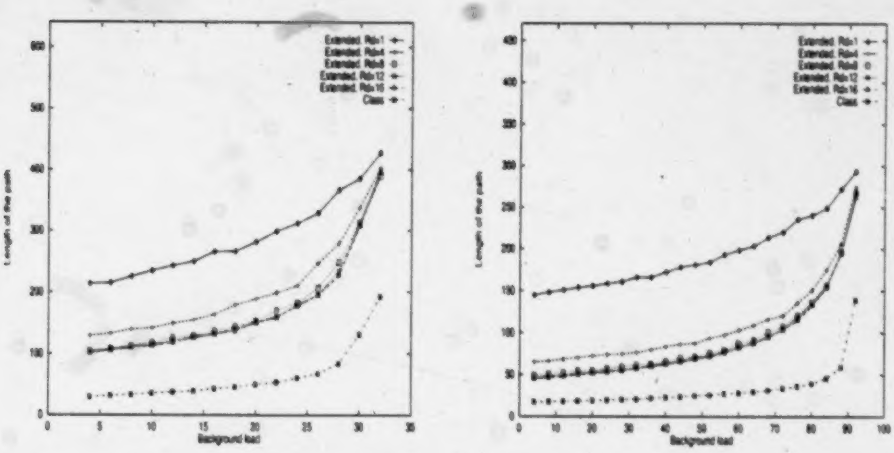
because first four destinations are closest to switch 0 through link A, while the remaining two destinations are closest to 0 through link B. Note that in fact, destination 17 could be carried in either of the two packets because the distance from 0 to 17 is 4 regardless of the outgoing link the fragment is to be routed through. In the case like this, the ultimate assignment of a destination to the replicated fragment is made at random.

3.6.3 Comparison of the Extended Schemes

As before, a new M-packet is sent only if the previous "main" multicast packet and *all* its replicated fragments have arrived to their destinations. The figures below show the delay results of R_Ext with different replication distances R_d compared to the performance of R_Class.

We can expect that the scheme replicating a packet into fragments that carry many addresses of destinations and/or increasing the probability that a packet will be replicated (R_Class or R_Ext with large R_d) will assure smaller delays than the scheme in which a fragment can be sent only to the immediate neighbour (R_Ext with small R_d). An overall path from the source to all D destinations will be usually shorter in R_Class than in R_Ext.

Generally, our reasoning is supported by the results of simulations presented in Figure 3.12. They show the path length vs the number of multicast destinations for different R_d in the R_Ext algorithm. They also show the performance of R_Class. The background



(a) k=2

(b) k=4

Figure 3.13: Delay vs background load for different replication distances R_d in R_Ext, and in R_Class

load in the network is 0.

Path length decreases significantly for R_d that increase up to the half of the maximum distance in the network (which is 17 for $k = 2$ and 16 for $k = 4$). Then, its decrease is more gradual.

When R_d is large, fragments of a given M-packet travel further from the switch in which they were replicated. This increases the probability of deflections caused by other fragments and that is why the length of the path does not decrease proportionally to the increase of R_d.

The combined path length is the shortest when R_Class is applied. Note that R_Class resembles R_Ext with the maximum replication distance. However, its performance is further improved by the fact that a replicated packet carries more than one destination address.

Figure 3.13 shows the path length vs background load for different replication distances R_d in the R_Ext algorithm, as well as the performance of R_Class. The number of multicast destinations is $D = 128$.

In R_Ext, when the background load is very high, the path length increases quickly and the results become similar regardless of R_d. Under such conditions, the number of replications is limited because outgoing links are more likely to be busy. Notably, the difference between the path length for $R_d = 1$ and for $R_d = 4$ is very large, particularly in the network with the higher connectivity. For higher values of replication distances, these differences become less significant.

The performance of R_Class is still better than the performance of R_Ext but also in this case, we observe a steep growth of the path length for high loads.

The schemes exploiting extended replication are only slightly more complex than their simple counterparts. For example, R_Ext with $R_d = 1$ is equivalent to R_All—every switch has to scan the list of destinations searching for the appropriate distance not greater than R_d. In R_Ext, however, this search does not stop on the first encountered distance that fulfills that condition. Instead, it continues in an attempt to find the minimum distance. The complexity of R_Class is similar. It only requires some extra time to create packets containing addresses of several destinations.

Clearly, the performance of R_Class exceeds that of R_Ext, and the performance of R_Ext exceeds the performance of the basic schemes with limited replication (Section 3.5). One might think that considering schemes based on limited replication is pointless, since they are worse than R_Class. However, there are two issues that must be considered before the selection of the best scheme for a particular situation.

Simplicity is the first issue. The complexity of routing decisions grows quickly with the increasing connectivity. At some point, the application of sophisticated multicast schemes may be too expensive. In that case, basic schemes (with or without limited replication) may be the only alternative. They offer a reasonable simplicity and an acceptable performance.

We have observed in figures 3.12 and 3.13 that in a network in which the number of replicated fragments is large (R_Ext with heavy replication, or R_Class), the performance of the algorithms may drop. It is particularly visible in the network in which the background load is high. A large number of fragments into which an M-packet may be replicated effectively increases the load in the network. These fragments may deflect other packets. They can also return to the source of the M-packet and throttle it. Note that in the above simulations, we assumed that the next M-packet may be sent only after all fragments of the previous M-packet reached their destinations. We have done this to be able to compare the multicast algorithms without the influence of other traffic. Obviously, this is not the case in "regular" networks in which M-packets are usually sent one after another. If each of these packets may be replicated into several copies, these copies may even further deteriorate the network performance. With the basic schemes, and even in the case of schemes with limited replication, this was not a problem—a copy of an M-packet could not deflect any other copy since it could be forwarded only to the immediate neighbour of the replicating switch.

In the next chapter, we study these two multicast schemes in the context of video applications. We will observe how strongly the application of a particular scheme affects the network performance.

3.7 Multicast Groups

As we already mentioned in Section 3.3, the size of the M-packet's header in the multicast algorithms presented in this chapter increases with the increasing limit on D—the number of multicast destinations. To avoid inflating the packet header without limiting the maximum number of multicast destinations too severely, it may make sense to divide a large deflection network into a number of *groups*.

Suppose that the maximum size of a multicast group is n and the number of multicast recipients turns out to be $m > n$. Upon the setup of a multicast session, the population of recipients can be logically divided into $G = \lceil \frac{m}{n} \rceil$ groups. A multicast source now has to send G copies of a single M-packet, each copy addressed to destinations within one group.

3.8 Summary

We have presented a number of simple multicast schemes for deflection networks. We have shown that with their application, multicasting in deflection networks is possible.

Some of the schemes use limited or extended replication, which do not depend on extra buffer space at a switch. The replication schemes outperform other schemes introducing relatively low overhead. Notably, among the simpler schemes based on limited replication, the one involving the least amount of processing (i.e., R_Random) turns out to be the best. In the network with $k = 4$, it is even better than the best basic scheme.

Among the schemes based on extended replication, R_Class is the best with the performance of R_Ext for large R_d being close to that of R_Class. However, we may expect that under some circumstances, the performance of these two schemes may be much worse than what we could expect. We will study this problem in the next chapter.

Still, if the complexity of the multicast algorithm is to be kept very low, the basic scheme B_Bounce is a good candidate, particularly in a network with low connectivity. Notably, among the schemes that require ordering multicast recipients only from the source, the least complex scheme B_Random is the best in the 4-connected network. In the 2-connected network, the chaotic path of a packet in B_Random makes this scheme only slightly worse than B_Furthest.

The proposed schemes seem to be especially well suited for relatively small multicast groups because the length of an M-packet's header increases with the increasing group size. This increase may be reduced by partitioning the multicast recipients—as suggested in Section 3.7.

In the next chapter, we investigate the performance of some of our schemes in high-speed applications like video-conferencing and transmission of video movies.

Chapter 4

Video Applications in Deflection Networks

We investigate experimentally the performance of deflection networks in several multimedia applications: videophone, videoconference and transmission of video movies. We show that despite the fact that deflection networks misorder packets and do not reserve any resources in advance, they may perform well in real-time applications.

In Chapter 2 (see also [OG98d]), we hinted on the possible suitability of deflection networks for jitter-sensitive, stream-oriented applications. We have observed that jitter and reassembly buffer space requirements in deflection networks are not very sensitive to changing patterns of the background traffic.

In this chapter we show that if the receivers are equipped with playout buffers of sufficient, and rather modest size, deflection networks may perform very well in "real-life" multimedia scenarios.

First, we present our simulation model. Then, in section 4.2, we show the simulation results of three video applications: videophone, videoconference and transmission of video movies. We use the fractal (self-similar) traffic model to generate synthetic traces. This model is believed to capture the behaviour of video sources [BSTW95]. We also propose a simple scheme for a resynchronization of the receiver and transmitter. In the context of a videoconference and a transmission of video movies, we apply and compare two of the multicast schemes proposed in Chapter 3. Finally, in Section 4.3, we conclude this chapter.

4.1 Network Model

Our simulation model resembles the one we used in investigating the jitter and reassembly buffers (Chapter 2). That is, we consider two network topologies: torus and triangle. Each switch is equipped with delay buffers whose purpose is not to store packets before they are forwarded, but to align packets arriving at the switch and to give the switch ample time to make a routing decision.

We have modified this model to be able to investigate the videophone traffic in deflection networks. In the further sections, it will be slightly altered to reflect videoconference and transmission of video movies.

There are two selected switches: $S0$ and $S1$. The distance between them is maximal in the network. $S0$ sends/receives packets only to/from $S1$. We investigate the performance of the session between the two selected switches under different background conditions in the remaining part of the network.

Every switch is equipped with a fractal (self-similar) traffic generator. An efficient algorithm for generating self-similar traffic was proposed in [RN96]. The traffic between switches $S0$ and $S1$ is described by Hurst parameter H, number of independent and probabilistically identical fractal renewal processes (FRPs) M, and load λ kB per frame. H specifies the degree of long-range dependence $(1/2 < H < 1)$, and M controls the burstiness of the model. For a more extensive study of this traffic model, see [BSTW95, GW94, LTWW94].

P other pairs of switches in the network may also exchange packets with load λ_b kB per frame. Hurst parameters in those sessions are uniformly distributed between H_{min} and H_{max}. After some time, P_c pairs are deactivated (i.e., their sessions are terminated), and new P_c pairs are randomly selected. Again, they send packets to each other with load λ_b.

Let N_P be the number of routing cycles (or packets passed through the switch divided by connectivity) per one frame time. This number is determined by the frame duration and by the network's capacity C. Every active switch counts the cycles. After N_P cycles, it invokes the traffic generator that returns the number of bytes. This number is converted to packets which are stored in the queue, and then transmitted as soon as possible, i.e., a packet is sent in the first free slot arriving from the network. Note that this way, it is possible that during one slot time, more than one packet will be transmitted. More specifically, a switch may send up to k packets during a single slot time (k is the network's connectivity).

When a packet arrives at its destination, it is removed from the network and the following performance measures are computed:

- throughput understood as the number of all packets (including those that were dropped because of their late arrival) that arrived to the receiver, divided by the time expressed in slots;

- access delay, i.e., the amount of time the packet spent in the sender queue;
- propagation delay from the moment of the packet's insertion to the network to the moment of its arrival to the receiver;
- playout delay in a playout (reassembly) buffer, that is, the time spent by the packet in the buffer until it is removed;
- end-to-end delay, that is, the sum of the access, propagation, and playout delays;
- packet loss, that is, the ratio of the number of dropped packets to the number of all packets that arrived at the receiver (including those that arrived too late)

Note that these measures are computed for both switches in the selected pair $S0$-$S1$ separately. When a packet is received by any other (background) switch, it is just removed, and only background throughput is updated.

Two selected switches $S0$ and $S1$ are equipped with playout (reassembly) buffers of size B frames (i.e., $B * N_P$ packets). At the beginning of a session, the playout buffers are empty. As soon as the first packet arrives to the receiver, the receiver starts the timer. After the time of $B_R = R * B$[1] , where $0 < R < 1$, it removes (receives) packets belonging to the first frame, and resets the timer. From this moment on, it will be removing consecutive frames from the buffer every frame time. Late packets that arrive after their frame has been removed from the playout buffer are dropped. Packets that overflow the buffer are also dropped. Note that this model of the reassembly buffer resembles the model presented in Section 2.2.6.

This approach is in fact equivalent to immediately moving of a frame to be played from the playout buffer to the device that actually displays the images on the screen. In fact, this process takes a few milliseconds [BO98]. Then, when a picture is ready to be displayed on the receiver side, it is inserted to the video frame buffer periodically scanned by the video adaptor to trace the image on the screen. This introduces additional, presentation delay of up to 17 ms [BO98]. As we will see in the next section, these delays do not seriously affect the end-to-end delays.

When a selected switch receives $3/2 * B$ frames, all its performance measures are reset. This warm-up period allows us to investigate the performance of an already active network.

---

[1] Note that since a frame has some duration determined by the capacity of the network and the number frames per second, we may actually use the size of the buffer to express time.

In the following sections, we present simulation results showing packet loss, end-to-end delay and throughput in several environments characteristic to videophone, videoconference, and transmission of video movies.

4.2 Results

Synthetic fractal (self-similar) traces used in the simulator were obtained from the traffic generator programmed on the basis of the algorithm given in [RN96]. Depending on the traffic characteristics, we adjust the load, Hurst parameter and the number M of FRPs so that the synthetic trace obtained from the algorithm will be close to the actual trace obtained from the analysis of a specific video scenario (e.g., videophone).

Simulator parameters common in all traffic scenarios are as follows:

- the network size is $N = 100$ switches;
- the fraction R of the playout buffer of size B is set to 0.8, that is, size of the active part of the playout buffer is $B_R = 0.8 * B$.
- all links are identical and their lengths are 1 packet (i.e., 424 bits);
- to obtain the maximum distance between the selected pair of switches, $S0 = 0$ and $S1 = 55$;
- every active source generates 24 frames per second;
- packet size is 53 bytes—48 bytes for data and 5 bytes for header (like in ATM);
- simulation time is 60 seconds;
- background traffic characteristics[2]
    - $\lambda_b = 10$ kB/frame;
    - Hurst parameter is uniformly distributed between $H_{min} = 0.60$ and $H_{max} = 0.70$. Values of H from this range are suggested to model the video traffic (see [BSTW95]). The number M that controls the burstiness of the traffic is set to 15. Video traces with this value of M most resemble the trace given in [BSTW95].
    - Every 2 seconds, $P_c = 2$ pairs are deactivated (two sessions are terminated), and new P_c pairs are randomly selected with new values of Hurst parameter. In fact,

---

[2] Note that they correspond to videophone traffic (see below).

we could expect rather longer sessions and, therefore, better performance. Note that frequent changes of the active pairs increase the burstiness of the overall traffic.

4.2.1 Videophone Traffic

The authors of [BSTW95] have shown the trace of a videoconference session. We assume that the trace of a videophone session would be similar—the only difference is the fact that in a videophone session, every user transmits/receives video to/from only a single user.

On the basis of this trace, we set the average load λ of the traffic between $S0$ and $S1$ to 10 kB per frame. Assuming that the frame rate is 24 per second, we obtain the average load of 1.97 Mb/s.

We set the Hurst parameter of the traffic between $S0$ and $S1$ to $H = 0.675$ which is within range 0.60 to 0.70. The number M of FRPs was set to 15.

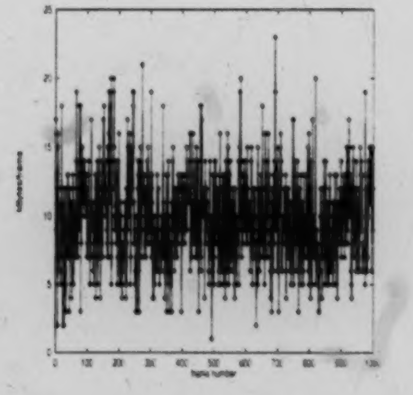


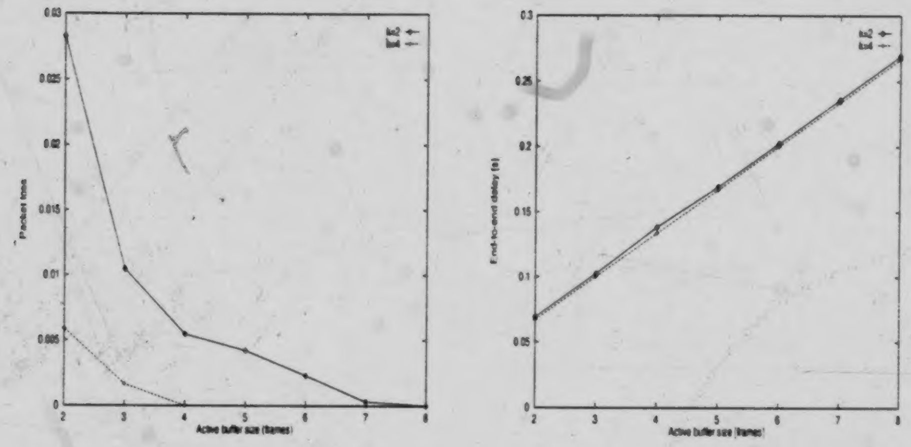
Figure 4.1: Videophone synthetic trace

Figure 4.1 shows a sample plot of bandwidth in kB/frame vs time for 1000 frames (cf. [BSTW95]).

We have performed simulations in two environments for two different connectivities:

- 2-connected network.

  There are $P = 34$ pairs of active background switches in the network which means that including the selected pair of switches, 70 switches (i.e., 70% of the entire network) take part in the videophone conversations. Network capacity is $C = 10$ Mb/s. Assuming that it takes a bit $5 * 10^{-9}$ sec. to travel 1 meter in the medium, with the link



(a) Packet loss

(b) End-to-end delay

Figure 4.2: Average packet loss and average end-to-end delay for both connectivities

length being equal to one packet, the physical distance between a pair of neighbouring switches is about 8.5 km.

- 4-connected network.
  In this environment, there are $P = 42$ pairs of active background switches. Network capacity is $C = 5$ Mb/s which, assuming links of 1 packet length, gives the distance of 17 km between the pairs of neighbouring switches. Other parameters are the same as those for the 2-connected network.

We will be changing size B of a playout buffer and investigating the throughput, average packet loss, and end-to-end delay in the network described above.

Figure 4.2 shows the relation between the packet loss and the active size of a playout buffer B_R, as well as an end-to-end delay. These measures are averaged over the two considered switches that take part in a videophone session. We may see a significant impact of the playout buffer size on the packet loss. For example, for $k = 2$, packet loss is very large (almost 3%) when $B = 2.0$ (i.e., $B_R = 1.6$). Then it quickly drops and for $B = 4.0$ it is less than 1%. For B greater than 7.0 no packets have been dropped. Notably, even if the playout buffer's capacity is $B = 8.0$ frames, its size does not exceed 8MB.

Of course, increasing the size of the playout buffer must increase the end-to-end delay. Figure 4.2b indicates that even for the largest buffer, this delay does not reach 400ms. Note that delays from 200ms to 400ms [WLS97] are allowed in this kind of application.

Throughput (i.e., number of packets received by the destinations per slot time unit) of a background traffic in the 2-connected network is about 14.90 and the sum of throughputs of the traffic between a selected pair of switches is about 0.45. The maximum throughput achievable in the above network is 18.76 thus we are looking at a network loaded at 81%. The 4-connected network is loaded at about 71% (the overall and maximum throughputs in the 4-connected network are 36.79 and 53.40, respectively).

Synchronization

Let us now modify the session model.

Packets are equipped with an additional field saying how many packets were generated during one frame time. The destination computes the percentage of packets from a given frame that were successfully received. If at least half of the packets coming from two consecutive frames were not received, the session is reset, that is:

- packets from the last received, garbled frame are dropped;
- the timer is set to the interval between the arrival time of the oldest and the newest packet, if this interval is not greater than B_R (expressed in time units) or B_R, otherwise;
- when the timer reaches B_R, the receiver resumes its normal functions, that is, it removes the next frame from the playout buffer, etc.

This model is likely to give better results than the previous one in many situations. Consider the following scenario. First, a few packets belonging to the first frame arrive at the receiver, which starts the timer that is supposed to trigger removing and playing the frames after time B_R. It implicitly assumes that packets will arrive at a more or less steady rate. However, it may happen that next packets arrive much later at the receiver because, for example, a new source on their path has become active and caused deflections. The receiver, unaware of this fact, will start playing at time B_R from the moment of arrival of the first packet. In this way, some packets will arrive too late at the receiver and, if its playout buffer size is not sufficient, a significant percentage of packets may be frequently dropped, particularly when the incoming frames are large. Of course, a similar scenario may also occur for frames other than the first one.

Assuming that the receiver works according to the new scheme, it will quickly "notice" that it is not synchronized with the transmitter. The above sequence of steps allows the receiver to resynchronize with the transmitter making the packet loss less frequent. Note that the receiver resets a session only when two consecutive frames of packets arrive badly garbled. In this way, the receiver does not reset (and drop packets) every time a single frame arrives garbled, which may be caused for example, by an unusually large number of packets transmitted during a frame time (e.g., a scene change).

Note that a loss of synchronization may be caused in another way. For example, packets belonging to the first half of the frame may arrive at a much smaller rate than the remaining packets. In such a case, if the playout buffer is short, the last packets may be dropped because the buffer is already full. A buffer overflow may also occur when a buffer contains several short frames, and then a long frame arrives. The buffer space freed after removing (and playing) a short frame may be insufficient.

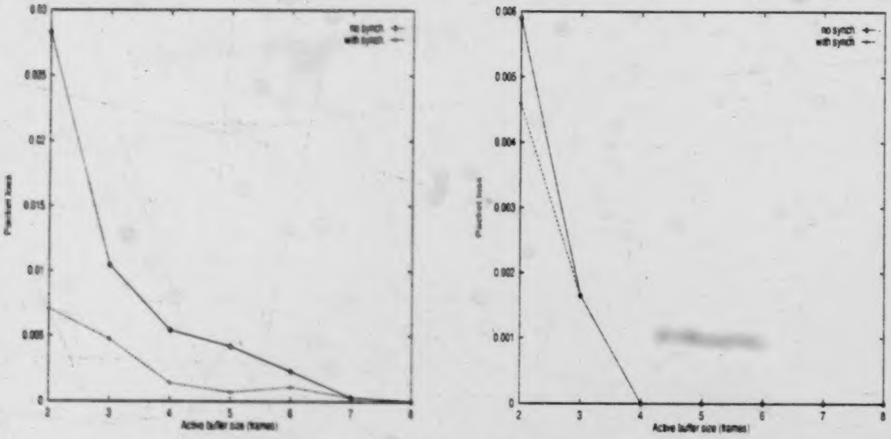
In the modified algorithm, when this situation occurs, the receiver drops $1 - R$ of the playout buffer starting from the oldest packets. For example, if $R = 0.8$, 20% of packets stored in the playout buffer will be dropped. This way, a part of the buffer that initially was supposed to be free, will be empty again.

Figure 4.3 shows a comparison between the packet loss and delay in both models. This scheme is particularly efficient when the buffer is much too small and packet loss is high. The likelihood of several garbled frames arriving one after another is high in such a situation and this is when occasional resynchronization helps most. When the traffic parameters of were set to obtain a very high packet loss (about 0.5), resynchronization could decrease this value even 10 times.

Note that the need for resynchronization usually indicates the inadequate size of the playout buffer. Thus, in a "real protocol", the process of resynchronization can be followed by increasing the buffer size.

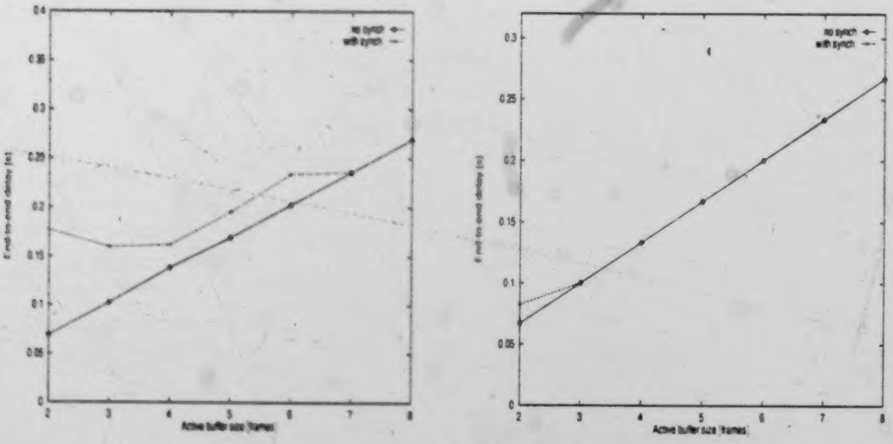
When packet loss is very small, like in the example network with $k = 4$, frames are rarely garbled sufficiently to trigger a reset. Under such conditions, the network works as if no resynchronization were applied. That is why, the plots of packet loss and delay converge for sufficiently big playout buffers.

In those cases in which the new model gives better results, end-to-end delays may be higher than in the model without synchronization. Simulations indicate that this is primarily caused by the increase in a playout delay (i.e., the time a packet spends in a playout buffer until it is removed). Note that when the receiver resets the session, the playout buffer may be empty. However, it is also possible that some packets belonging to the frames other than the one that triggered the reset are there. These packets may wait



(a) Packet loss, k=2

(b) Packet loss, k=4



(c) End-to-end delay, k=2

(d) End-to-end delay, k=4

Figure 4.3: Average packet loss and end-to-end delay—comparison of synchronized and asynchronized versions of the algorithm

longer for their turn to be removed and played which is especially visible for $k = 2$ and $R = 3$. Notably, even under these conditions, the end-to-end delay does not exceed the recommended values.

Our subsequent results were obtained from the model with resynchronization.

4.2.2 Videoconference Traffic

The model presented in Section 4.1 is now altered to reflect a videoconference traffic.

There are A randomly selected, videoconference sources/receivers. Every switch in this pool acts like a source that transmits multicast packets to the remaining $A - 1$ switches, and like a receiver that receives multicast packets transmitted by $A - 1$ sources. Traffic and network parameters (load, Hurst parameter, synthetic trace, capacity, etc.) for a given connectivity are the same as those for the videophone traffic (Section 4.2.1). This time, however, there are:

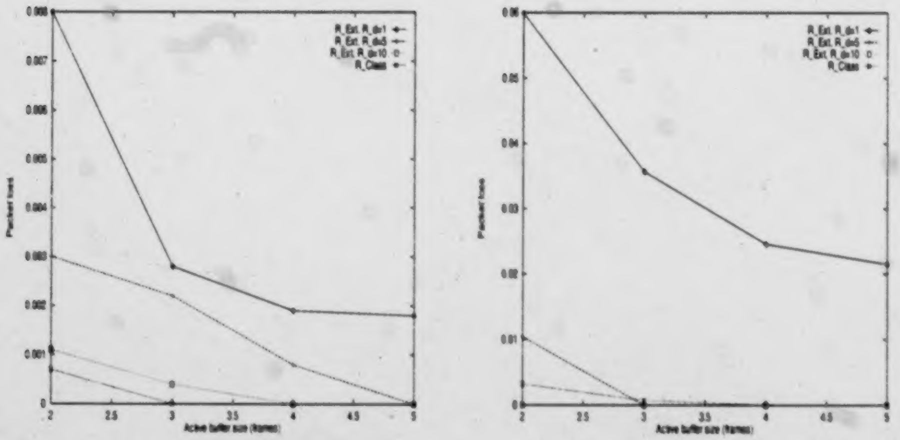
- $P = 25$ pairs of background switches involved in a videophone traffic in the 2-connected network, and:
- $P = 28$ pairs of background switches in the 4-connected network.

Along with $A = 4$ videoconference switches, it gives 54 active switches in the 2-connected network, and 60 active switches in the 4-connected network. Only the model with resynchronization has been applied.

We will compare the performance of two multicast algorithms (schemes) and their suitability in a videoconference application. The generic multicast algorithm was described in Section 3.2. The two multicast algorithms: extended ***R_Ext*** and class ***R_Class*** were proposed in Section 3.6.

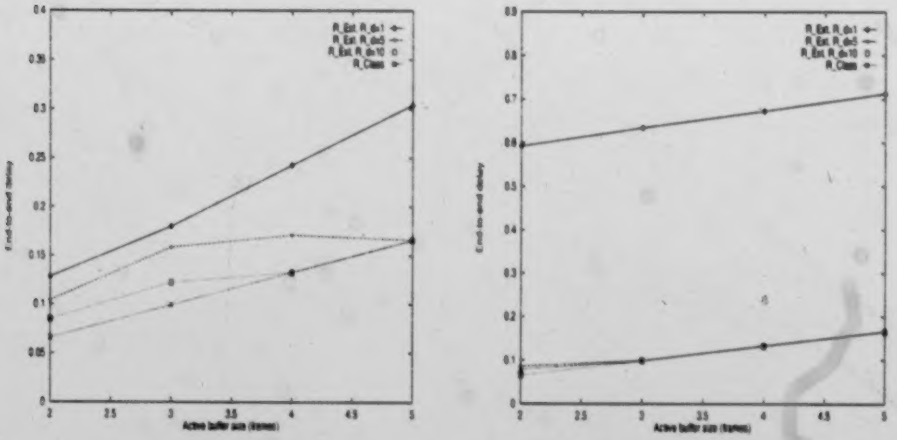
At first sight, we should expect that the scheme replicating a packet into fragments that carry many addresses of destinations and/or increasing the probability that a packet will be replicated (R_Class or R_Ext with large R_d) will give better results. Since an M-packet with all its fragments will arrive faster to the destinations, it should less interfere with other packets, thereby decreasing the packet loss.

Our reasoning is supported by simulation results presented in Figure 4.4. They show the average packet loss and the average delay between the participating switches. R_d has three values: minimum, medium and maximum. Note that the diameter of the torus (maximum distance between a pair of switches) is 10 so defining $R_d > 10$ has no sense.



(a) Packet loss, k=2

(b) Packet loss, k=4



(c) End-to-end delay, k=2

(d) End-to-end delay, k=4

Figure 4.4: Average packet loss and end-to-end delay in videoconference

The smallest packet loss is observed when R_Class or R_Ext with large replication distance is applied. Note that R_Class somewhat resembles R_Ext with the maximum replication distance. Its performance may be further improved by the fact that a replicated packet carries more than one destination address.

Delay relations between particular algorithms are similar to the relations observed in Figure 4.4a. R_Ext with large R_d, or R_Class assure a shorter average path (and propagation delay). Since packets arrive to their destinations faster, they are less likely to block the source and suffer deflections from other packets—this, in turn, decreases access delay. Moreover, packets arrive at destinations more regularly and resets are less frequent. We have already explained that session resets may increase playout delays (for example, for R_Ext, $R = 2$ and $R_d = 1$, sessions were reset four times while for $R_d = 5$ only one session was reset). It all adds up to smaller delays when schemes R_Class and R_Ext with large R_d are applied.

Resetting a session may have a strong impact on the playout delay and thereby on the average end-to-end delay. We can see a trade-off here—if the algorithm without resynchronization were applied, the delays would be more predictable and perhaps lower but a packet loss would be very high.

The throughput of the background traffic in the 2-connected network is about 10.99 and the sum of throughputs of the traffic among the videoconference sources/receivers is about 0.88. The network is thus loaded in 63%.

Notably, only delays of the worst algorithm (R_Ext with $R_d = 1$) are too large for this kind of application. For other replication distances, and particularly in R_Class, zero packet loss was obtained when the size of the playout buffer was sufficiently small to assure an acceptable end-to-end delay.

4.2.3 Video Movie Traffic

The model is further modified to reflect transmission of video movies.

There is only one selected source $S0$ and a number D of randomly selected destinations (receivers). The source transmits multicast packets addressed to D destinations, according to either of the two multicast algorithms described in the previous subsection.

No traffic originates at the destinations. P pairs of other background switches are activated. Videophone sessions between them were described in Section 4.2. As before, no performance measures except the throughput are computed for the background traffic.

The authors of [GW94] have shown the trace of an example video movie. On the basis of this trace and data given in that paper, we set the average load λ to 26 kB per frame. Assuming that a frame rate is 24 per second, we obtain the average load of 5.11 Mb/s. This load is within the limits of 3 – 6 Mb/s given in [DBH96].

We also set the Hurst parameter of the traffic originating from the transmitter to $H = 0.8$, as suggested in [BSTW95]. The number of FRPs was again set to 15.

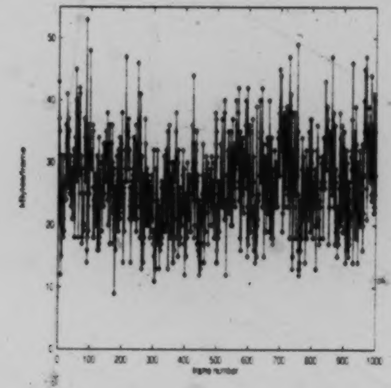


Figure 4.5: Video movie synthetic trace

Figure 4.5 shows the sample plot of bandwidth in kB/frame vs time for 1000 frames.

We set the number of destinations to $D = 30$ and network's capacity to $C = 10Mb/s$. We have performed simulations in two environments for two different connectivities:

- 2-connected network.

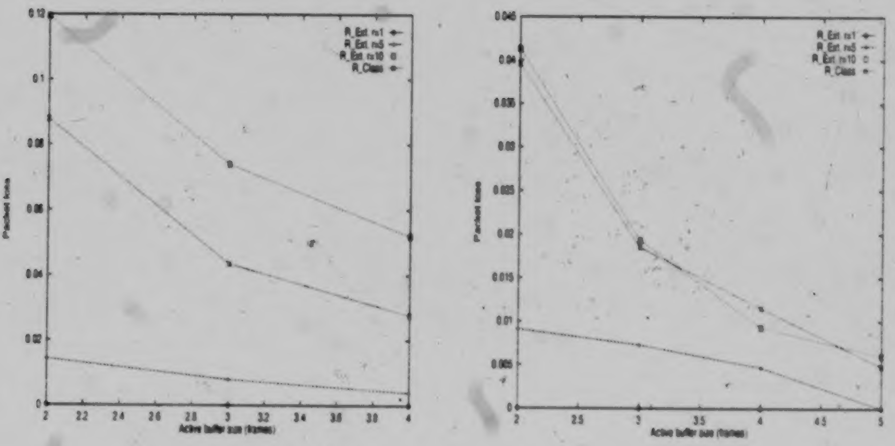
  There are $P = 5$ pairs of active background switches in the network. It means that, including D video receivers and one video transmitter, 41 switches (i.e., 41% of the entire network) are active.

  Other parameters are the same as those in the videophone model.

- 4-connected network.

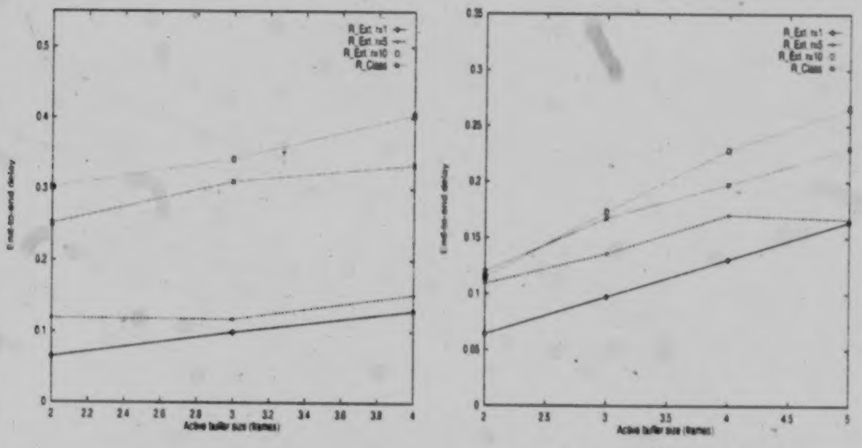
  The number of pairs of active background switches is $P = 25$. Including D video receivers and one video transmitter, 81 switches (81% of the network) are active.

Figure 4.6 shows the average packet loss and end-to-end delay in the network described above. Surprisingly, the relations among the results for different multicast algorithms are completely unsimilar to what we have seen before. R_Ext with replication distance $R_d = 1$ gives almost zero packet loss. With increasing R_d, packet loss becomes more significant. It



(a) Packet loss, k=2

(b) Packet loss, k=4



(c) End-to-end delay, k=2

(d) End-to-end delay, k=4

Figure 4.6: Average packet loss and average end-to-end delay in transmission of video movies

is worst when R_d is maximal. Scheme R_Class that was giving the best results in videoconference, performs very poorly in transmission of video movies.

In Section 3.6.3 we have signaled that this behaviour is possible. Due to the fact that every replication in R_Class may produce several M-packets that will further divide in subsequent switches, a single M-packet sent by the source may actually produce many multicast fragments that will circulate in the network. These fragments may inhibit not only other switches but the source of the M-packet itself.

We observe a similar phenomenon in R_Ext when the replication distance is maximal. The main multicast packet sent by the source may be fragmented into many packets (up to $D = 30$) whose paths may be very long. These packets are likely to suffer many deflections and they also may return to their source blocking it from transmissions.

Our observations indicate that indeed, the source in the 2-connected network transmitting according to R_Ext for $R_d \doteq 10$ is blocked by its own packets more often than in R_Class. Specifically, the ratio of packets blocked at the source by its own packets to the overall number of successfully transmitted packets is 1.62 in R_Ext for $R_d = 10$ and 1.46 in R_Class. This causes a significant increase of access delay in R_Ext.

Due to the fact that in R_Class (or R_Ext with high R_d) the source may be more frequently blocked because of the higher load, and that packets suffer more deflections, both access and propagation delays are high. It also makes receiver resets more likely which, in turn, may increase playout delays. It all adds up to the generally higher end-to-end delays when R_Class or R_Ext with higher replication distances are applied.

The situation was quite different in the videoconference application because the number of destinations in that case was only four. The number of multicast fragments produced by R_Class or R_Ext with a large replication distance could not be high.

However, when we deal with large multicast groups, like the one considered in the transmission of video movies, the number of fragments a given M-packet divides into may become too large. In other words, this algorithm (as well as R_Ext with large replication distance), may overly increase the network load. When the load is high, the performance (packet loss and delays) deteriorates. Therefore, when the number of multicast receivers is high, it may be beneficial to limit the number of fragments circulating in the network by setting R_d to small values. Note that $R_d = 1$ is always safe because fragments may be relayed only to immediate neighbours—they cannot be deflected so they cannot increase the load. The basic schemes, or schemes based on limited replication presented in the previous chapter would also give better results than R_Ext with big R_d. Scheme R_Class should be rather avoided in applications in which a number of multicast destinations is large.

Delays obtained in this scenario are acceptable. The constraints imposed on the end-to-end delays in the transmission of video movies are much less severe than videoconference constraints. For example, the authors of [DBH96] suggested a playout buffer holding about 800 ms of video stream. This is even more than our overall end-to-end delay.

On the average, the throughput between the source and a single receiver is about 0.50 (it naturally depends on the applied scheme). The throughput of the background traffic in the 2-connected network is about 2.22. The overall throughput between all sources and receivers is thus about 16.59 which means that the network is loaded in about 88%.

4.3 Summary

We have studied the performance of three multimedia applications in deflection networks. We have also proposed an efficient and simple synchronization scheme that significantly reduces packet loss, particularly when the playout buffer is too small or when the conditions in the network temporarily disrupt the regularity with which packets should arrive at the receiver. We have examined the suitability of two multicast schemes for videoconference application and transmission of video movies.

Our results suggest that despite the fact that packets may arrive at the receivers misordered, the performance of deflection networks in multimedia applications may be good. Note that the maximum network capacity used in the above simulations was only 10 Mb/s for the 2-connected network, and 5 Mb/s for the 4-connected network. This is less than, e.g., the 100 Mb/s capacity of Fast Ethernet. It is also *much* less than gigabit speeds at which deflection networks are expected to work. Obviously, with the increasing capacity of the network, we can only expect the improvement of the performance of the investigated applications.

To see why, let us notice that the number of slots passed through the switch during the time of one frame equals

$$n = \frac{C}{L * f}$$

where C is the network's capacity, L is the packet (or slot) length, and f is the frequency at which video sources generate frames. We may view n as a bandwidth that may be used by the source to insert its own packets to the network. Obviously, with the increasing number of active sources and/or increasing load of the particular session, this bandwidth will shrink due to the higher fraction of busy slots in n.

Clearly, with increasing capacity C, available bandwidth n increases proportionally to increasing C. For example, with capacities reaching 100 Mb/s, we obtained zero packet loss with small playout buffers sizes.

It is interesting to notice how the performance of a given multicast scheme may vary in two different applications (transmission of video movies and videoconference). Assuming that a videoconference does not involve tens of users, we may be certain that the application of R_Class or R_Ext with large replication distance will give the best results. However, in case of transmission of video movies, the number of receivers may reach hundreds in a large network. Since the performance of e.g., R_Ext with big R_d may be disastrous in one case (many users) while it may be best in another (a few users), a source could switch from one multicast scheme to another depending on the size of the multicast group. In case of R_Ext this operation would be very straightforward—an M-packet may carry the value of R_d in its header and an intermediate switch would use this value in the routing decision. A similar change from R_Ext to R_Class is also conceivable.

The packet loss at a given moment may differ depending on the switch being monitored. This is caused by the distribution of the active switches, which may be more or less beneficial for the investigated source. It may happen that some source is surrounded by active background switches whose packets frequently inhibit its packets. Higher interarrival delay and irregularity with which packets arrive at the receiver cause a bigger packet loss.

If we increase the network load, allowing more sources to be active, the performance measures will certainly deteriorate. The receivers will need larger playout buffers to accommodate the higher jitter. The large buffers, in turn, will increase the end-to-end delays, perhaps exceeding the values allowed for a given application.

For this reason, a protocol assuring that the required quality of service will be sustained throughout the whole session is needed. This protocol would have to limit the number of active sessions in the network. For example, the network may be heavily loaded when a source intends to initiate its session. In such a case, if the required quality of service (e.g., packet loss and end-to-end delay) cannot be delivered, the source should be notified about it and its session should be blocked.

In this chapter, we wanted to show that even with a limited network capacity and fairly high load, a deflection network is capable of achieving packet loss and delays acceptable in certain standard multimedia applications. The results suggest that if we could only overcome the problems mentioned above, deflection networks could be suitable for multimedia applications.

Some attempts aimed at limiting datagram sources for the sake of isochronous sources in deflection networks are presented in the next chapter. Finally a protocol providing a quality of service in a deflection network is presented in Chapter 6.

Chapter 5

Techniques Giving a Preferred Treatment to Isochronous Traffic

Providing service guarantees in a deflection network is difficult. There is no easy way to reserve the resources, due to the multitude of paths that may be taken by a packet belonging to a given session. On the other hand, the high-speed network should possess the ability to provide and sustain the required quality of service which is necessary in real-time applications.

Several solutions to this problem have been proposed in the literature. For example, backlogged switches may signal their needs using a dedicated field in the packet header [BF92]. In response to those requests, other switches may refrain from transmission and return tokens that provide the backlogged sources with required access to the bandwidth. Another solution [MLNP93] is based on imposing a virtual multi-ring topology onto the physical (mesh) topology and using a fair bandwidth allocation scheme suitable for rings. With that scheme, the performance of connection-less service for bursty traffic suffers quite significantly. Moreover, it assumes that the network is in fact an MSN, i.e., its physical topology is that of a torus and its connectivity is 2.

Feedback-based solutions (e.g., using tokens) are generally poorly scalable, because they tend to waste bandwidth in proportion to the bandwidth-delay product of the network. They also increase the complexity of the protocol and counteract the whole idea of deflection by trying to imitate some aspects of synchronized networks.

In this chapter (see also [OG98c]) we investigate how far one can get with synchronous traffic in deflection networks using a combination of the following simple measures:

- Prioritized routing for synchronous traffic in those cases where the standard routing rules are randomized (Section 5.2).

- Age-driven preference rules aimed at containing the maximum delay and jitter (Section 5.1).

- Statistical ways of throttling unconstrained data traffic at its sources, to avoid preempting synchronous sources (Section 5.3).

We propose a protocol implementing these measures, and investigate it under several types of traffic conditions ranging from Poisson to bursty. With the application of this protocol, excessive datagram traffic is throttled, and the performance of isochronous traffic is sustained regardless of the intensity and type of the datagram traffic.

5.1 Network Model

The network model resembles the one we used in investigating the jitter and reassembly buffers (Chapter 2). That is, we consider two network topologies: torus and triangle. Each switch is equipped with delay buffers whose purpose is not to store packets before they are forwarded, but to align packets arriving at the switch and to give the switch ample time to make a routing decision. We assume that the total delay involved in a single hop in the network is the same for all links and equal to a single slot.

There are two kinds of traffic in the network: datagram (data), and stream (isochronous). The stream model is as follows. Some number of randomly selected switches transmit isochronous packets to the selected switches at rate R expressed in $slots^{-1}$. For example, the fixed arrival rate of 1/3 means that a new packet is generated in each stream source every three slots. Such a packet will be expedited in the first free slot arriving from the network. The stream source generates packets at a fixed rate. The remaining, datagram part of the network generates packets according to some distribution. If competing packets have the same age, one of them is chosen at random and deflected.

In this study, we use the *age* routing strategy [HRL95]. If two packets of the same kind (i.e., either stream or datagram) prefer the same outgoing link, the one that has made more hops so far is given its preferred link. If stream packet competes with a datagram for the same outgoing link, it wins with probability P. This is the way in which a stream traffic can be given priority over datagram traffic.

Age deflection gives worse average routing decisions than, for example, locally optimal routing [BC90] in which packets are assigned to outgoing links such that the sum of the distances to their destinations is minimized. However, it assures that the maximum delays suffered by stream packets are lower. This fact also suggests that the jitter (as defined below) may be lower. Indeed, the results of simulations (not given here) in which these two routing strategies are compared seem to confirm this conjecture.

Let the number of packets in the queue of a stream source S at some moment equal Q. If S accumulated $Q + A_v$ packets in its queue and has not sent any packet since the number of packets in the queue was equal to Q, then the stream session at switch S is:

- *aborted* if $Q > 0$.
- *blocked* if $Q = 0$.

In either case, the source clears its queue and starts the transmission from scratch.

Our objective is to monitor the throughput of stream and datagram traffic, as well as the jitter and the number of blocked and aborted sessions of stream traffic.

5.2 Prioritized Routing for Synchronous Traffic

In this section, we show how varying probability P of giving a stream packet priority over a datagram affects the throughput and jitter of synchronous traffic.

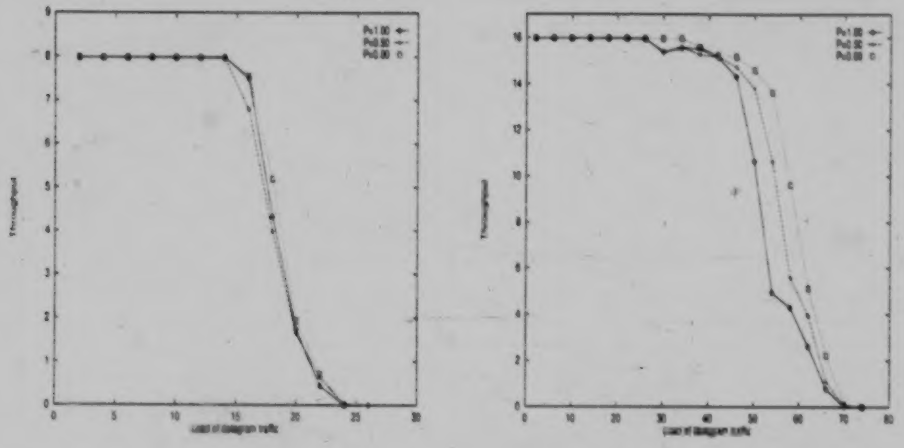
If $P = 1.0$, the age routing algorithm works as follows. Stream packets are considered first. They are ordered by decreasing age and then, beginning from the oldest one, assigned to the "best" outgoing links. Then, datagrams are considered. They are similarly ordered and assigned to the remaining links.

If $P < 1.0$, every stream packet competing at a switch may temporarily "become" a datagram with probability $1 - P$. That is, it will not be considered in the first phase of the algorithm. Instead, it will compete with datagrams as if it were a datagram itself.

Intuitively, the best results should be obtained when $P = 1.0$. In such a case, stream packets have always the better chance to be routed over their preferred ports. They are always given priority over datagrams.

However, the results showing the observed throughput of the stream traffic vs datagram load (Figure 5.1) are quite surprising (throughput of datagram traffic is not shown). The simulated network is a torus with 256 switches, and its connectivity (the number of link pairs per switch) is 2, and 4.

For low datagram loads, the throughput is similar regardless of P. Stream packets in such conditions rarely have to compete with datagrams and the value of the probability is rather irrelevant. The situation changes when the load gets heavier. The throughput is highest when the probability P that a stream packet is given priority over a datagram is **lowest**. It is particularly visible in the 4-connected network.



(a) k=2

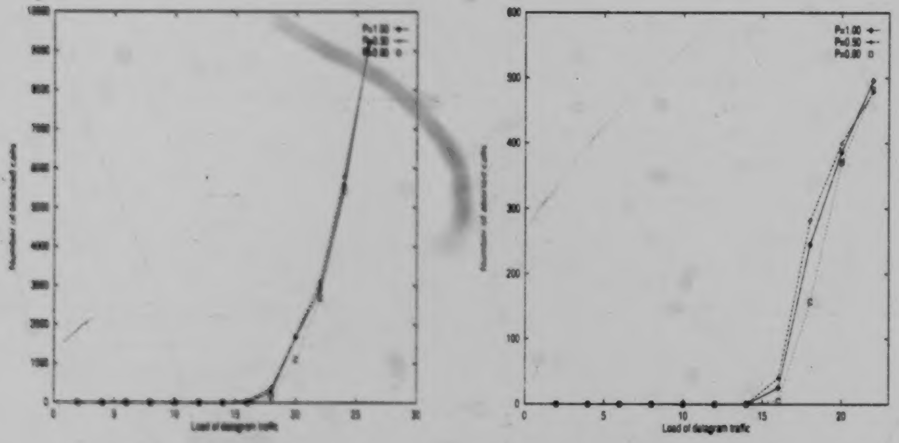
(b) k=4

Figure 5.1: Throughput under uniform Poisson datagram traffic for different probabilities P, datagram traffic contribution 0.16, and rate of stream sources 1/10

To find out why this happens, let us look at Figure 5.2 showing the numbers of blocked and aborted sessions vs datagram load. For high loads, calls are blocked and aborted less frequently for lower probabilities P. It seems that if stream packets follow less predictable paths (i.e., $P < 1.0$), they block other stream sources less often. This is understandable: when stream packets always win the contention with datagrams, it is more likely that they will follow the same paths to their destinations. It also implies that other stream sources located on such paths will be more likely to have problems with regular access to the bandwidth. It means that sessions originating from such switches will be blocked and aborted more often.

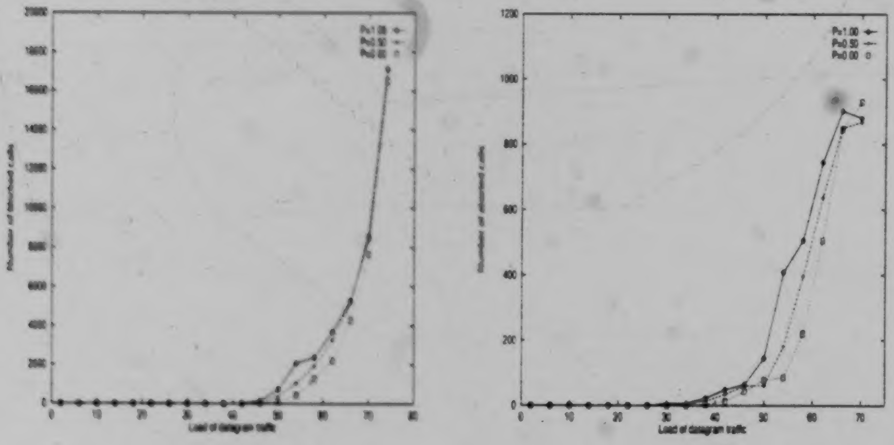
This effect is less visible for the lower connectivity because packets simply have a smaller routing flexibility. It means that many packets are bound to follow similar paths and possible randomization does not change much.

However, there is a side-effect of improving the throughput by increasing randomization in the network (i.e., decreasing P). Figure 5.3 shows jitter of isochronous sessions vs datagram load in the above environment (the method of jitter computation was explained in Section 2.1). It may be noticed that jitter in the network in which stream packets have always priorities over datagrams ($P = 1.0$) is several times lower than jitter for $P < 1.0$. Since with decreasing P the quality of routing decisions worsens (from the point of view of a stream packet), a packet on average will need more time to arrive to the destination.



(a) blocked calls, k=2

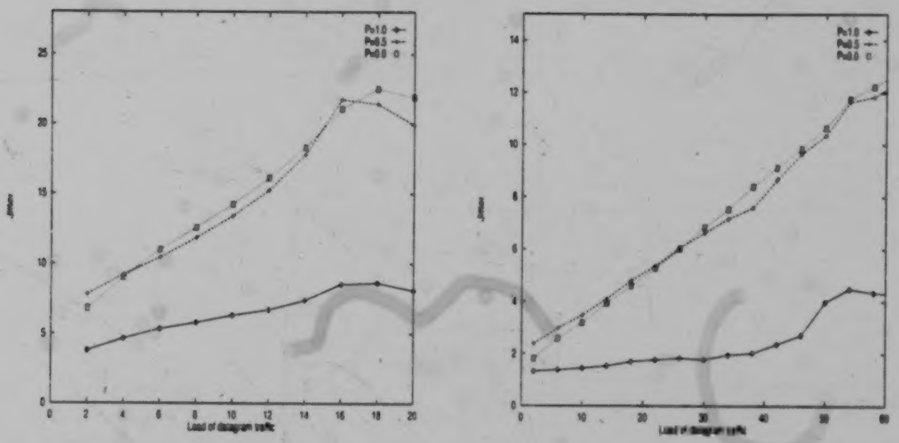
(b) aborted calls, k=2



(c) blocked calls, k=4

(d) aborted calls, k=4

Figure 5.2: Number of blocked and aborted calls under uniform Poisson datagram traffic for different probabilities P, datagram traffic contribution 0.16, and rate of stream sources 1/10



(a) k=2

(b) k=4

Figure 5.3: Jitter under uniform Poisson datagram traffic for different probabilities P, datagram traffic contribution 0.16, and rate of stream sources 1/10

The randomization of routing decisions and paths that helped to achieve better throughput now increases the unpredictability of time intervals at which stream packets arrive at the destination. This in turn significantly increases the jitter.

We have made similar observations for P close to 1. The throughput deteriorates and the jitter improves with increasing P. The above observations show that the randomization in deflection networks improves at least some of the performance measures. However, small values of P should be avoided if jitter is to be contained, that is, if we deal with, for example, video transmissions. Probably, values close to but less than 1 would be best as they cause minimal deterioration of throughput and decrease the probability of deadlock.

Plots of the number of aborted calls and jitter are shown only for loads under which not too many sessions are blocked. This is because for the highest loads, the jitter decreases. This artificial effect comes from the fact that under high load, a small number of sessions take part in jitter computation. Many sessions are aborted or blocked, and those that make it to the end of the experiment must come from the few sources which, because of their fortunate locations, are not seriously disturbed by datagrams.

We can limit the number of stream sessions in the network (and limit the throughput) but assure that these sessions experience a low jitter, or we can admit more sessions to the network (and increase the throughput) but significantly increase the jitter. In the first case, [illegible] $= 1.0$ while in the latter case, $P < 1.0$. As we have mentioned, the low jitter is very important in isochronous applications. For this reason, in the following sections, we will consider the age routing algorithm in which probability P that a stream packet is given priority over a datagram is 1.0. This way, we will achieve the lowest possible jitter for the price of a decreased throughput.

5.3 Throttling Datagram Traffic

It is obvious that even with priorities assigned to isochronous packets, the stream traffic will suffer for sufficiently high loads of datagram traffic. Under heavy load, datagrams will occupy most of the slots. It will make the access of stream sources to the bandwidth less frequent and predictable. It will also increase the number of blocked and aborted sessions. One way of solving this problem is to throttle excessive datagram traffic when it becomes too high, giving stream sources more opportunities to transmit their packets. The algorithm exploiting this observation is as follows.

Let every source of datagram packets be equipped with token counter T_c, let factor $f = k/2$ (k is the network's connectivity), and let $0 \leq T_c \leq T_{c_{max}}$. If switch S senses b busy slots (a slot carrying a packet addressed to S is considered busy) in the current routing cycle, it performs:

$$T_c = T_c + f \times (k - b) \tag{5.1}$$

if $b \leq \frac{k}{2}$, or

$$T_c = T_c - f \times (b - \frac{k}{2}) \tag{5.2}$$

if $b > k/2$, where $f = k/2$.

A switch, or rather a host attached to it, may send a datagram if $T_c > k$. Every transmission decreases T_c by k down to the minimum of 0. The maximum value of the token counter, $T_{c_{max}}$, is initially set to $2 \times k^2$—it allows the switch to accumulate tokens for transmission in two cycles of slots.

Every time a datagram switch acquires m packets to transmit in one cycle, it performs:

$$T_{c_{max}} = T_{c_{max}} + k \times (m - m_{old})$$

if $m > m_{old}$, or

$$T_{c_{max}} = T_{c_{max}} - k \times (m_{old} - m)$$

otherwise. m_{old} is the previous value of m, that is, non-zero number of datagrams that the switch acquired in some previous cycle in which m was positive. After adjusting the maximum number of tokens, the switch increases the token counter:

$$T_c = T_c + f \times m \qquad (5.3)$$

In all cases, T_c must not be greater than $T_{c_{max}}$, and $T_{c_{max}}$ must not be less than $2 \times k^2$.

Let us analyze the benefits of this algorithm. If the network load is light, and incoming slots perceived by a switch are usually free, the switch can freely transmit its packets. If the load is heavy, the switch cannot transmit its datagrams every time it sees a free slot. Instead, it will have to refrain until its token counter T_c returns to the appropriate value (i.e., at least k), relaying free slots to other switches.

To see how this mechanism works, let us analyze the following example in which the network connectivity $k = 4$, the source accumulated $T_c = 12$ tokens, and $Q = 10$ packets. Since every transmission decreases T_c by k, the source has permissions to transmit 3 packets. Let us assume that the load is high and the number of busy slots in several consecutive cycles is $b = 3$ per cycle (i.e., there is only one free slot per cycle). In every cycle, the token counter will be decreased by 2 (see Formula 5.2), and additionally by 4 because of packet transmission in the available free slot. This way, after the transmission of two packets and the time of two cycles, T_c will drop to 0 and transmission of S will be stopped. From now, it will pass the single free slots it perceives in subsequent cycles. Note that this source will regain the access to the bandwidth only if it acquires new packets to transmit (Formula 5.3), or if more free slots start passing through the source (the load will get lighter).

Now, let us assume that the load is moderate and the number of busy slots in several consecutive cycles is $b = 2$ per cycle. Assuming the initial assumptions ($T_c = 12$ and $Q = 10$), the token counter in every cycle will increase by 4 (see Formula 5.1), and decrease by 8 because of transmission of two packets in both available slots. After the time of three cycles, and the transmission of six packets, T_c will drop to 0. From now on, the source will send one packet per cycle because T_c will increase by 4 according to Formula 5.1, and immediately drop to zero because of transmission of a *single* packet. Thus, the source will be filling only one slot per cycle, passing the other one to the downstream switch. This will throttle the source allowing other sources (particularly the stream ones) to access the network bandwidth.

When a switch acquires a single burst of packets, it increases the token counter to enable the transmission of half of that burst, and adjusts the maximum value of the token counter to accommodate the number of tokens that will eventually allow the switch to send all packets in the burst. Thus in subsequent cycles, if it only receives the sufficient number of free slots, it will transmit the whole burst. If the source continues to acquire the packets in bursts of the same size, the token counter will increase but its maximum value will remain the same. This way, the switch will not be able to accumulate too many tokens, which will inhibit the datagram traffic.

The fact that $T_{c_{max}}$ changes with the number of packets acquired by a switch in a single cycle makes it possible to achieve better performance when the traffic is bursty. At the same time, it does not deteriorate uniform traffic in which packets appear in the switch at similar rates, i.e., when the datagram traffic is uniform.

Our scheme resembles the token bucket used to shape the traffic generated by bursty sources. In that scheme, the tokens are placed (with some rate) in the bucket, which has a limited capacity. If the bucket fills, newly arriving tokens are discarded. To transmit a packet, the system must remove a token. Thus, similarly to our mechanism, a source cannot send a packet if it does not have any tokens. However, if it only has enough tokens, it can send a burst of packets limited by the maximal number of tokens it can collect (the bucket capacity).

The most important difference between the token bucket and our mechanism is the fact that the number of available tokens in the token bucket is not affected by the current load perceived by the source. In our scheme, the value of the token counter depends on how many free and busy slots pass through the source. The transmission rate of the source is in fact regulated by the background load. In the extreme case of the very high load, the value of the token counter may drop to zero blocking the transmission from a source. Clearly, this mechanism could not be used in isochronous sources that rely on the timely transmission of their packets. However, the token bucket could not be used in our deflection network, because it is unable to limit the transmission rate of the source in case of the high load.

We may thus view our scheme as a version of the token bucket in which the number of tokens (and the source activity) is a function of a network load.

Instead of our statistical approach to limiting the datagram load in the network, we could use another approach used in local and metropolitan area networks. In this approach, a source would be allowed to send its packets only if it possessed a token. Note that the meaning of this token is different from what was previously discussed. In the case of the token bucket, a token was a permission to send a packet internally generated by the source. Here, a token is understood as a special packet that circulates in the network granting the sources that capture it access to the bandwidth.

This approach has several limitations even in broadcast networks it was intended for (see Chapter 1). Token networks do not scale well and tokens themselves waste network resources.

Our simulations have shown that if we allowed only one token to circulate in the deflection network, the throughput would be significantly lower than that of a network without the token access. Imagine, for example, a network consisting of 256 switches in which only one source at a time can transmit its packets.

We could allow multiple tokens to circulate in the network. The possession of a token would not have to be a necessary condition for a transmission [Kam90, CS92, DG93]. This approach works well in the networks in which we can predict the path of a packet. They do not work in deflection networks because we simply do not know what path to the destination the packets will choose. Packets can be deflected at any switch and block the sources that posses a token.

It seems that some known approaches that work in store-and-forward or broadcast networks do not work in deflection networks. That is why we have to resort to application of other schemes like the one we have presented above.

We will show in Section 5.4 that this scheme prohibits the excessive datagram traffic to accumulate in the network, allowing stream switches to access the required amount of bandwidth. In this scheme, a stream traffic has always priority over a datagram traffic and the latter one is statistically throttled.

5.4 Results

In this section we present some of our simulation results obtained for a torus network implementing the throttling mechanism described above. The results for the irregular, triangle network were very similar and they are not shown here. The stream traffic has always priority over datagram traffic (i.e., $P = 1$—see Section 5.2), and the latter one is throttled (Section 5.3). The network has 256 switches, and its connectivity (the number of link pairs per switch) is 2, and 4. The value of $A_v = 10$ packets. We will investigate the throughput, jitter and the number of blocked/aborted sessions for different contributions c $(0 \leq c < 0.5)$ of the stream traffic vs datagram load. c determines how many source-destination pairs take part in the stream traffic. Every switch is equipped with a queue of size $k/2 * 2500$. This queue stores packets that cannot be sent in a given cycle.

Note that if a datagram traffic is not throttled, the throughput of a stream traffic will decrease with increasing load of a datagram traffic. In such a case, stream sources will be overflown with datagrams and their sessions will often be aborted. It may also happen that a high intensity of datagram sources will make the queues of these sources to overflow.

Therefore, a simulation is run for increasing datagram load and it terminates if either the queue of any datagram source overflows or the throughput of the stream traffic drops below 0.01.

The routing scheme handles datagrams differently. Namely, stream packets are routed according to the age algorithm but datagrams are routed according to the locally optimal algorithm. We have mentioned that the latter scheme offers better throughput but worse jitter. Since we are not interested in the jitter of datagram traffic, we use the better routing scheme to handle it.

5.4.1 Poisson Datagram Traffic

This scenario was described in Section 2.2.1. In this case, however, only datagram sources transmit their packets according to Poisson distribution.

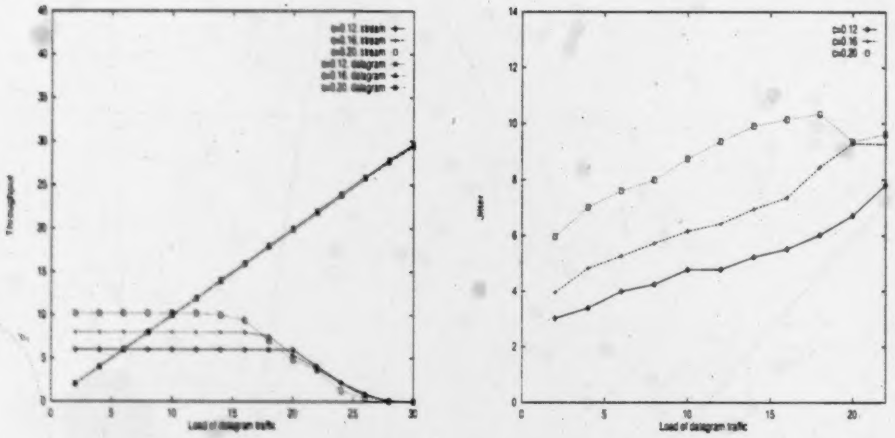
Figure 5.4 shows the stream throughput and jitter vs datagram load for different contributions of the stream traffic and different connectivities. The token counter is not used, i.e., the datagram traffic is not throttled. However, the isochronous stream traffic is prioritized.

For some range of datagram loads, giving priorities to the stream traffic keeps its throughput at the same level, regardless of the datagram load. However, as the network becomes more saturated with datagrams, these packets "overflow" stream sources not allowing them to transmit at the required rate. More and more sessions are aborted, and eventually the stream throughput drops to 0. The datagram traffic is not affected.

As expected, higher *c* makes this range shorter because of the relatively greater load in the network. The jitter increases slightly with the increasing datagram load because datagrams cause more deflections and the access of stream sources to free slots is less regular. We have observed similar relations between the datagram load and the jitter of stream sessions under all datagram traffic scenarios. For this reason, this performance measure is omitted in the next subsections.

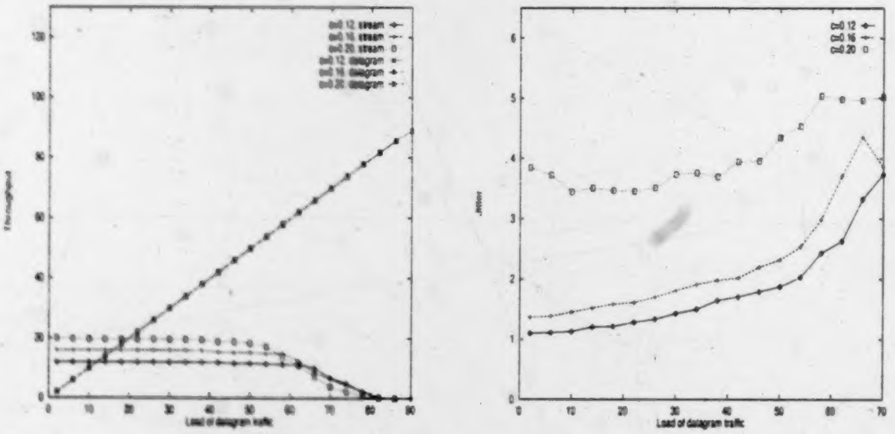
Figure 5.5 shows the stream throughput vs datagram load for a given contribution of stream traffic and different connectivities. The plots allow us to compare two protocols: the one that uses the token counter, i.e., throttles excessive datagram traffic, and the one that only gives priorities to synchronous traffic.

When the token counter is used, the throughput of the stream traffic remains constant for a wide range of datagram loads, then it deteriorates very slightly. The throughput of



(a) Throughput, k=2

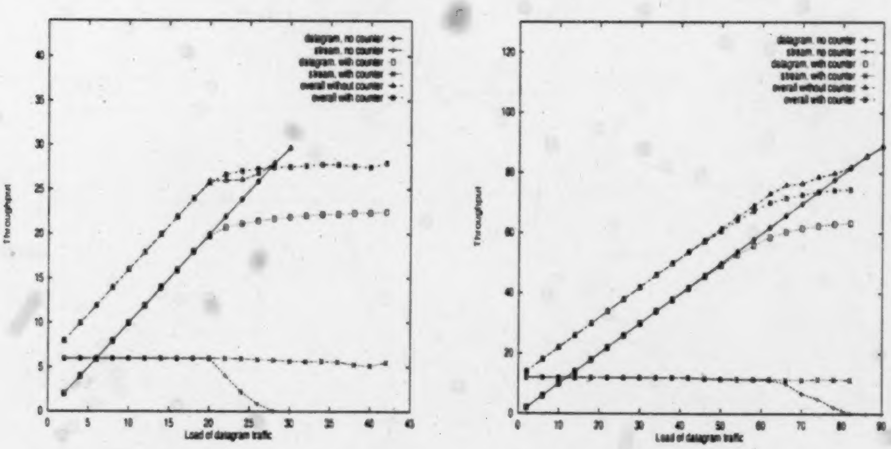
(b) Jitter, k=2



(c) Throughput, k=4

(d) Jitter, k=4

Figure 5.4: Throughput and jitter under uniform Poisson datagram traffic for different contributions c of stream traffic and transmission rate of stream sources is 1/10



(a) Throughput, k=2

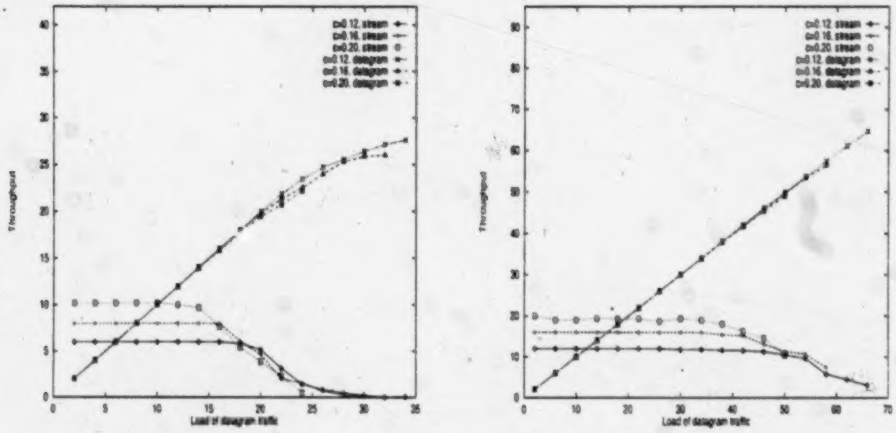
(b) Throughput, k=4

Figure 5.5: Throughput under uniform Poisson datagram traffic for contribution $c = 0.12$ of stream traffic and transmission rate of stream sources 1/10

the datagram traffic drops in comparison to the case when no token counter is used. The fact that the overall throughput (datagram + stream) is almost the same in both protocols indicates that the applied token mechanism is efficient. At the same time, no bandwidth is wasted for negotiations.

The jitter (not shown) is low and its increase is somewhat slower and more regular when the token counter is used. The difference between its values for low and high datagram loads is small. For example, in the 2-connected network, this difference is about 5 slots. The increase is from the jitter of 2.5 slots for zero datagram load to almost 8 slots for the highest datagram load of 42 packets per slot time unit. These values are even smaller in the 4-connected network. The small values of jitter and its relatively low sensitivity to the increase of datagram load is important particularly in jitter-sensitive applications like voice or video. We have observed similar relations between the jitter and the load under other datagram traffic scenarios described further in this section.

We have also observed a large difference in the number of blocked and aborted sessions (not shown). In the uncontrolled network, unconstrained datagram sources often inhibit stream sources forcing them to abort or block their sessions. In the controlled network, datagram sources are not allowed to overly increase their load and the number of blocked/aborted sessions is almost zero. It all assures that the quality of service offered to isochronous users will remain constant regardless of the intensity of a datagram traffic. We



(a) Throughput, k=2

(b) Throughput, k=4

Figure 5.6: Throughput under correlated Poisson datagram traffic for different contributions c of stream traffic, transmission rate of stream sources 1/10, $P = 0.4$, and frequency of pair changes $l = 1000$

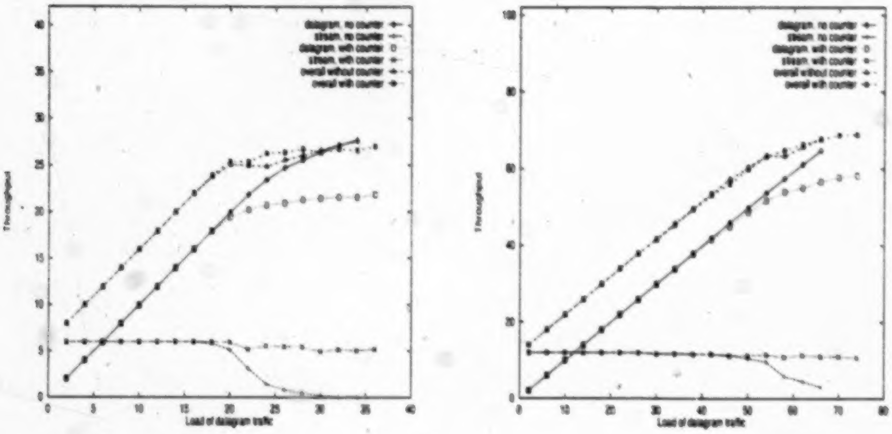
have observed similar relations for other patterns of the datagram traffic.

Plots for higher contributions of stream traffic $c = 0.16$, $c = 0.20$ and larger are similar to those shown in Figure 5.5. That is, for some load, datagram throughput starts to decrease, while the stream throughput remains almost the same.

5.4.2 Correlated Poisson Datagram Traffic

This scenario was introduced in Section 2.2.2. Again, only datagram sources transmit their packets according to this distribution. The results of the simulations in which the datagram traffic was not throttled are shown in Figure 5.6. They are very similar to the results presented in the previous subsection.

In the 4-connected network, simulations terminate not because the throughput of the stream traffic drops below 0.01 (like before) but because the queues in datagram switches overflow. The queues in datagram sources in both cases (i.e., for both Poisson datagram traffic patterns) are the same, but under correlated traffic, the switches tend to acquire packets in bursts rather than uniformly. Recall that n sources transmit their packets with a greater probability that the remaining part of the network. Although these groups change every l cycles, those sources that appear in consecutive groups are more likely to accumulate more packets in their queues.



(a) Throughput, k=2

(b) Throughput, k=4

Figure 5.7: Throughput under correlated Poisson datagram traffic for contribution $c = 0.12$ of stream traffic, transmission rate of stream sources 1/10, $P = 0.4$, and frequency of pair changes $l = 1000$

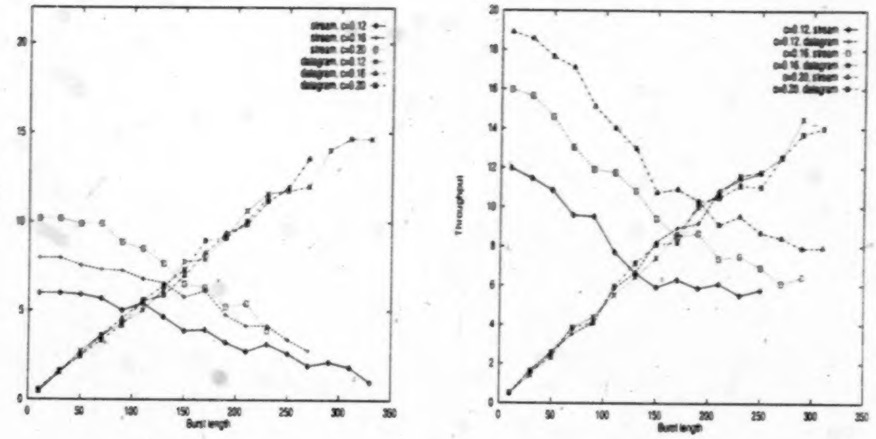
The results of experiments shown in Figure 5.7, are very close to those presented in Figure 5.5. That is, in the uncontrolled network, an increase in the datagram traffic causes a gradual deterioration of the stream throughput. In the network in which datagram sources are throttled, the throughput of stream sources remains stable, and the overall throughput is even slightly better than in the case with no token counter. As before, the relations between the datagram load and jitter (not shown) are similar to the relations observed in the previous subsection.

This also indicates that deflection networks are not sensitive to the changing patterns of loads, at least for as long as the global load in the neighbourhood remains at the same level.

Plots for higher contributions of stream traffic (not presented here) $c = 0.16$ and $c = 0.20$, and for less frequent changes of pairs $l = 2000$ and $l = 4000$ are again similar to those shown in Figure 5.7.

5.4.3 Bursty Synchronous Datagram Traffic

A bursty traffic scenario was described in Section 2.2.3. As before, only datagram sources transmit their packets according to this distribution. The simulation results of the uncontrolled network are shown in Figure 5.8. Again, the relations between the throughput for



(a) Throughput, k=2

(b) Throughput, k=4

Figure 5.8: Throughput under bursty datagram traffic for different contributions c of stream traffic, transmission rate of stream sources 1/10, and $t = 20$

both traffic types are similar to what was shown in previous subsections.

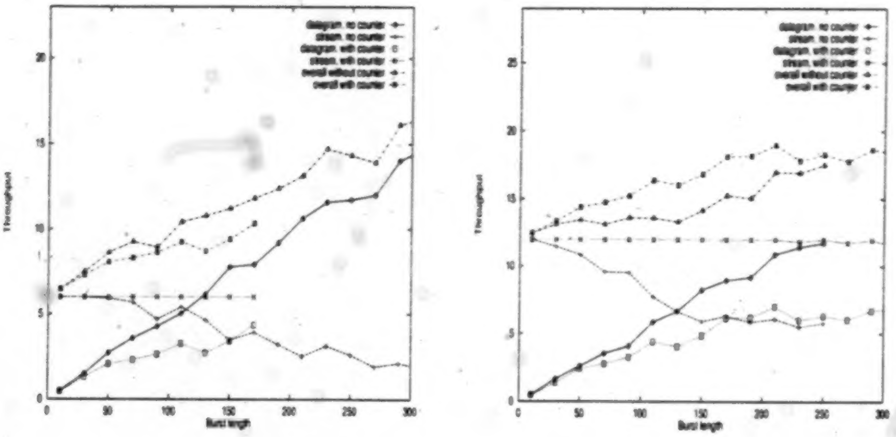
Similar to the correlated Poisson traffic, datagram queues overflow before the stream throughput reaches 0.01. This phenomenon is even more pronounced here.

The results comparing the two protocols are shown in Figure 5.9. It seems that the token mechanism works well even under this traffic scenario. As before, the throughput of the datagram traffic decreases while the stream throughput remains almost the same. However, a more serious deterioration of the overall throughput may be observed.

Plots for higher contributions of stream traffic $c = 0.16$ and $c = 0.20$ (not presented here), and for different sleep times $t = 10$ and $t = 30$ are again similar to those shown in Figure 5.9.

5.5 Summary

We have presented a protocol implementing simple measures aimed at providing isochronous service for stream traffic by throttling the datagram traffic. We investigated the performance of our protocol under several types of traffic conditions ranging from Poisson to bursty. The simulation results indicate that the network performance for synchronous and isochronous traffic remains stable, even under relatively heavy datagram loads. The performance for datagram traffic decreases, but the total decrease in the maximum throughput (assuming



(a) Throughput, k=2

(b) Throughput, k=4

Figure 5.9: Throughput under bursty datagram traffic for contribution $c = 0.12$ of stream traffic, transmission rate of stream sources 1/10, $t = 20$

uniform load) is small.

The increase in jitter is also small which suggests that our protocol is useful in jitter-sensitive applications. The protocol works efficiently even when the network topology is irregular (triangle) and a large fraction of all stations participate in synchronous sessions. Our solution does not require any pre-reservation of resources, and no bandwidth is wasted on token exchange.

The main drawback of our solution is its unfairness. If a switch A overflows its less active neighbour B with data traffic, then B will have to refrain from transmission rather than A. Maxemchuk [Max91] has shown how to achieve fairness in the deflection network. His solution involves collecting information as to the bandwidth requirements of every switch along the path of a packet. This information is written to the packet's header. Upon reception of the packet, the destination checks if the transmission requirements of some switches are greater than the available number of slots passing through them. If so, the destination sends a separate request to the transmitter asking it to decrease its transmission rate.

The drawback of this solution is the end-to-end delay. It may happen, particularly in high-speed networks, that before a source learns that it has to decrease its transmission rate, some other source will get blocked, and will have to abort its transmission. This solution also does not consider isochronous sessions whose specific timing constraints cannot be exceeded.

However, our algorithm could be easily merged with this technique for achieving fairness in a deflection network. With this approach, our algorithm would quickly respond to an excessive datagram load, while the slower fairness enforcement protocol would guarantee long-term fairness.

Regardless of the adopted policing schemes, there is always some boundary on the combined contribution of the stream sources to the network load. Our simulations show that if the transmission rate and the number of stream sources increase to a certain point, the stream sources, whose traffic is not inhibited by the algorithm, will overflow each other with packets. This will compromise the performance of stream traffic even in the total absence of a datagram load.

Thus, our next approach will involve regulating the stream traffic that enters the network (Chapter 6). We will see that by limiting the load and adjusting the reassembly buffers, we can achieve a long-term quality of service.

Chapter 6

Service Guarantees in Deflection Networks

As we have indicated in the previous chapter, control over the quality of service in a pure deflection network is rather difficult. A source connected to a switch may send its packet only if at least one incoming slot is empty. In a heavily loaded network, most of the incoming slots are non-empty and the source may not be able to transmit at the required rate. This may have a negative impact on delay- and jitter-sensitive traffic.

The network may be heavily loaded when a source intends to initiate its session. In such a case, if the required quality of service (e.g., packet loss and end-to-end delay) cannot be delivered, the source should be notified about this and its session should be blocked. It is also possible that an already active source is suddenly overflown by packets coming from adjacent sources initiating their sessions.

In the previous chapter we have shown that throttling the datagram traffic, and some other simple measures, may help the network to sustain a given throughput of synchronous traffic. That protocol, however, was still unable to control the admission of synchronous sessions and to provide a long term quality of service.

In this chapter (see also [OG]), we propose a protocol equipped with mechanisms allowing an already initiated session to sustain its quality of service regardless of other activities in the network. We focus on a videophone application, although this protocol can be easily extended to accommodate applications like videoconferencing and transmitting video movies (see Section 6.5).

6.1 Network Model

The network model used in this chapter resembles the model used for investigating the performance of deflection networks under video traffic scenarios (Chapter 4). That is, every switch is equipped with a self-similar traffic generator. The parameters of this generator are set such that the video trace corresponds to the trace observed in a videophone application (Section 4.2.1).

All switches are equipped with fractal (self-similar) traffic generators described by Hurst parameter H, the number M of independent and probabilistically identical fractal renewal processes (FRPs) [RN96], and the load of λ kB per frame. H is uniformly distributed between H_{min} and H_{max}. Every specified number of seconds, P_c new random pairs of sources are activated in the network.

As before, the switches are equipped with playout (reassembly) buffers of size B frames. The purpose of these buffers is to smooth out jitter and order possibly misordered packets. C will denote the network's capacity.

A fractal traffic generator at every source generates packets which are then stored in the transmitter queue. From this queue, they are removed and inserted to the network at some rate. This time, the transmission rate is not the attribute of the source, but it is determined on the per outgoing link basis. That is, a transmission rate from a source may be different depending on the outgoing link. More precisely, the process of inserting packets to the network is as follows.

First a routing decision at switch S assigns incoming packets (i.e., packets in transit) to outgoing links. Then, if any outgoing links are free and if the packet queue at S is non-empty[1] , S inserts its own packets. For example, assume that a transmission rate through link 0 is 1 and through link 1 is 1/2. S will try to send its packet over link 0 if only that link is free, and will use no more than every second free slot perceived on link 1.

Initially, the transmission rate at the given source is maximum and it is set to the same value $1/r_l$ for all links l ($0 \leq l < k$, where k is the network's connectivity), and $r_l = 1$. That is, the source is allowed to insert its packets to the network in every available free slot. Note that it is possible that during one slot time, more than one packet will be transmitted. More specifically, a switch may send up to k packets during a single slot time.

Also, we will no longer look at the performance of a session between one selected pair of switches only. We will observe how the increasing network load affects the number of blocked and active pairs, throughput, packet loss, and delay in our protocol.

---

[1] To simplify the description, we will not differentiate between a switch and the host connected to it.

6.2 The Protocol

In a videophone session, there are two parties S and $\hat{S}$. Source S sends/receives packets to/from the corresponding sender (peer) $\hat{S}$. Thus, S and $\hat{S}$ must act as both transmitters and receivers.

Before sources S and $\hat{S}$ start a videophone session, they transmit dummy packets to each other. Load and other parameters of this traffic must be "similar" to what a transmitter can expect in the session in which "real" packets carrying video data will be transmitted. During this time the pair of sources determines if the network is capable of processing their traffic with the desired quality of service. This phase of the transmitter protocol will be called *setup*.

The setup phase may complete only when the pair of sources have determined whether their tentative session does not disturb the already active sessions, and the quality of service as perceived by the receivers at S and $\hat{S}$ is fulfilled. Upon a successful completion of the setup phase, the transmitter will begin the normal session—this phase will be called *active*.

It is possible that a new source will not be allowed to begin the active phase, which is caused by the fact that not enough resources are available in the network to provide the required service. In that case, the session is blocked.

We start the description of the protocol from the active and setup phases of the transmitter. Then, we describe the way the receiver accepts or rejects a connection based on the required quality of service.

6.2.1 Active Phase

A video source generates n_f frames per second. If the network's capacity is C and packet length is L, a source may insert up to $\frac{C}{L * n_f}$ packets during the time of one frame. The size of the given frame, i.e., the number of packets generated during the one frame time, also depends on the transmission load and the current activity of the source. Every frame is first put to the queue from which packets are removed and inserted to the network at some rate local at every outgoing port.

F_Q, also called the *observation period*, is the number of frames that serve to compute the average size of the packet queue right before the generation of subsequent frames.[2] Namely, before a new frame i is generated, the source stores the current size of the queue Q_i. Ideally, this size is zero—number of free slots passing through the source during the time of one frame should be sufficient to transmit all packets from that frame. After F_Q frames are generated, the source checks F_Q last values of Q_i. On the basis of their values, the source determines if the queue has been growing (see below). If it has not, the source resumes the observation period.

---

[2] Note that since a frame has some duration determined by the capacity of the network and the number of frames per second, we may actually use the size of the buffer to express time.

Let V_g denote the number of times when $Q_{i+1} > Q_i$, and $V_l = F_Q - V_g$ show how many times $Q_{i+1} \leq Q_i$ for $i = 0..F_Q - 1$, where Q_i is the size of the queue right before frame i is generated.

The queue is considered to be growing within the observation period n, if $V_g^n \geq V_l^n$ and $Q^n_{F_Q-1} > Q^{n-1}_{F_Q-1}$. In other words, the queue is considered to be growing if:

- the number of packets added to the queue within the current observation period is usually greater than the previous one; and
- the size of the queue at the end of the observation period is greater that the size of the queue at the end of the previous observation period.

In this way, the example sequence of Q_i: 0 30 25 18 15 will not indicate the queue's growth. This sequence may represent temporary increase of the queue size caused by a long burst of packets generated by the source.

If the queue size has increased, it means that most likely, packets coming from other sources have been inhibiting source S. Generation of packets at S is faster than their transmission. In that case, S finds the setup source K (i.e., the new source in a setup phase) such that:

$$V = M_K^l * \frac{1}{r_l + 1} \tag{6.1}$$

is maximized. M_K^l is the number of packets per frame sent by the setup source K through its link l that passed through S. $1/r_l$ is the transmission rate of K through link l. Clearly, values of r_l, l as well as the status of the source must be carried in all transmitted packets.

If K slowed down its transmission rate from $1/r_l$ to $1/(r_l + 1)$, Formula 6.1 would approximate the factor by which the number of packets transmitted by K through link l would drop. Packets transmitted through this link often pass through switch S and that is why S will try to slow down K's transmission over this link.

Now, the transmitter has to check if the condition

$$\gamma_Q - V < 0 \tag{6.2}$$

holds, where γ_Q is the number of packets added to the queue per frame averaged over the last F_Q frames. If this holds then, assuming a similar activity of the setup source K in the future, the decrease of the number of packets transmitted by K through link l should stop the growth of S's queue.

It may happen that the above formula will not hold for only one value of M_K^l. This would suggest that slowing down only one source will not help. In this case, S will determine the next source K that gives large value of V. The new value of V will be added to the old one and again S will check if the above condition holds.

If condition 6.2 holds, S will send a *slow-down* (SD) request[3] to source K. Note that if more than one source were determined by the above algorithm, sending that request could be implemented as a multicast packet. The source will now refrain from transmission of other SD messages (even if its queue grows) for the time of 3 frames. As will become clear in Section 6.2.2, this time is sufficient to make sure that an inquired source K had enough time to decrease its transmission rate or block the session. After that time, the number of packets from K passing through S should decrease sufficiently. Source S will find out about it when it resumes counting F_Q frames, and measuring the sizes of its queue Q_i. If, after another observation interval, S determines that the queue still grows, another SD request will be sent. S will keep sending SD messages to setup sources until its transmitter queue ceases to grow.

If the queue at S were likely to grow (according to Formula 6.2) even if all sources registered during the last F_Q frames would agree to slow down their transmission rates, S may send SD to some sources several times. It is thus possible that a given setup source will decrease its transmission rate over some link more than once on the request of a single source. If a setup source is unable to do it without the increase of its queue, the session it initiates will be blocked.

Generally, it is possible that for some time, the packet queue at S will grow making the arrival of packets to the receiver less regular. It still does not have to cause a deterioration of service because the receiver may adjust its playout buffer to accommodate the higher jitter (see Section 6.2.3).

As a rule of thumb, we set F_Q to 4. If F_Q were greater, the source would react slower to changes in the network. For example, it would require more time to send SD message responding to the packets coming from the new source.

---

[3] All acronyms of control messages are listed in appendix A.

It becomes clear why we insist on separating transmission rates through particular links. Imagine the following scenario in which the transmission rate at the source is global.

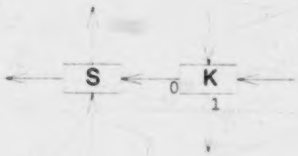


Figure 6.1: Separating of transmission rates

On the slow-down request from S, the 2-connected setup source K slows down its transmission rate from 1 to 1/2. Clearly, S would like K to slow down its activity on link 0 (see Figure 6.1). From now on, K will pass every second slot free. However, we do not know which slots will be free, those passed over link 0 or over link 1. It may happen that most slots will be passed on the other link and the number of packets from K passing through S will not change. In that case, slowing down the other source simply does not work.

This phenomenon is particularly harmful in a 4-connected network. The probability that a source will pass free slots over links different than the required one is 3/4 as opposed to 1/2 in a 2-connected network.

On the other hand, this issue may lose its significance if S is very far from K. In that case, it is more likely that packets sent through different outgoing links will arrive to S from the same incoming link particularly in a 2-connected network.

Let us now consider the following scenario. Sources S and K are active, and source L has just begun the setup phase. L's packets do not pass through S but they often deflect K's packets such that they go through S. If S's queue grows, S should send an SD message to L. However, it cannot do it because L's packets do not pass through S which does not know that L started the setup phase. If no other active source blocks L, L will eventually begin the active phase, and the quality of service of the session from S may degrade. Clearly, L's session should be blocked or slowed-down because it caused problems in an already active source. However, in this model, S has no way to block the new source L.

For this reason, every deflection caused by setup, data packets is recorded. Namely, when a packet sent by source L in the setup phase deflects a packet sent by an active source S, the L's Id is recorded in the header of the deflected packet. Every new deflection overwrites this information which means that only the last deflection is recorded. Those deflecting packets are treated as if they directly passed through S. That is, they are considered when the transmitter at S needs to send a slow-down request.

Thus, in the above scenario, S will send the SD request to the "offending" source L despite the fact that its packets have never directly inhibited S packets. This will result in blocking source L.

When a source has nothing more to transmit, it simply sends a *connection finished* (*CF*) message to the receiver.

6.2.2 Setup Phase

In this phase, source S and its peer $\hat{S}$ send "dummy" packets trying to determine if the session between them is likely to sustain the required quality. For some time, S monitors the behaviour of its queue. If the queue grows, it usually means that the network load is high and S is often inhibited by packets coming from other sources. In such a case, the session being initiated by S is blocked. The session is also blocked when the peer $\hat{S}$ is blocked, when the quality of service, as perceived by the receiver at $\hat{S}$ is not fulfilled (see Section 6.2.3), or when the source receives a slow-down SD request (see Section 6.2.1) with which it cannot comply.

First, let us see how the setup source S responds to a slow-down request. The source that receives SD request from switch K checks if:

$$\tau + \frac{1}{r_l} * \tau_l < N_F \tag{6.3}$$

where:

- $1/r_l$ is the S transmission rate over link l. The purpose of the SD request sent by K is to slow down the transmission rate at S over this link.
- τ is the difference (in time) between the moment of transmission of the last packet from the frame and the moment at which the frame was generated. This value is averaged over the last F_Q frames.
- τ_l has a similar meaning to τ but it concerns link l. That is, τ_l records the time between the moment of frame generation and transmission of the last packet from

this frame *through link l*. Note that $\tau = Max(\tau_l)$ for all links $0 <= l < k$, where k is the connectivity.

- N_F is the time needed to pass one frame of packets. It depends on the network's capacity and frame duration.

If S slows down its transmission rate from $1/r_l$ to $1/(r_l + 1)$, it will take on average $\frac{r_l+1}{r_l} * \tau_l - \tau_l = \frac{1}{r_l} * \tau_l$ more time to send the same number of packets through link l. When this value is added to τ, the result stands for the overall time S needs to send all the generated packets. If this value exceeds N_F, S's queue will be likely to grow because the time of one frame will not suffice to transmit all packets generated during this time. In this case, the session initiated by S will be blocked. If condition 6.3 holds, S will decrease its transmission rate through link l and resume the observation period. In this period, S will determine if slowing its transmission rate does not really make its queue size to increase, and if it is enough for the active source to no longer be overflown by S packets.

The setup time may be expressed as:

$$T_{setup} = v * I * F_Q$$

I is the minimum number of observation periods of F_Q frames that must elapse before the new source may determine if its session may or may not be accepted (see below). $I * F_Q$ gives a new source enough time to make sure that its activity does not disturb active sources.

However, it may happen that during this time, the receiver and the corresponding sender will fail to notify S if the session (from their perspective) may be admitted. It is also possible that S will slow down its transmission rate upon request of some active source and will need more time to determine if the new conditions (i.e., lower rate—see above) will allow it to carry on and if the source that requested the slow-down is satisfied. In all the cases, S will begin another observation period of $I * F_Q$ frames. Thus, S will repeat this process v times.

T_{setup} is variable because v is variable. It is affected by:

- the round trip propagation delay;
- the fact that both S and its corresponding sender $\hat{S}$, as well as the receiver at $\hat{S}$, may interrupt the session at any time of their setup phases; and
- responses to slow-down messages issued by active sources.

I should be at least 3. Consider the following scenario. Let us assume that source S is in the active phase, and source K is in the setup phase. Let us also assume that K started to overflow the active source S with its packets. If S started to perceive K's packets e.g., in the middle of its ith observation period of F_Q frames, those packets might not have inhibited its own packets severely enough to justify a transmission of an SD message to K. However, after the next, $i+1$ observation period, S's queue may increase forcing this source to send SD to K. Thus, it may take $2 * F_Q$ frames for S to realize that a new source K disturbs its session. Considering the propagation delay of SD sent to K, it becomes clear that $I \geq 3$. During the time of $I * F_Q$, a source in the setup phase will learn if its packets disturb other sessions.

We set I to a "safe" value of 5. It may happen that the first "dummy" frames transmitted by a setup source are not large. That is, the number of packets transmitted by a new source may be too small to seriously disturb active sources. Larger value of I gives active sources more time to block such a source.

Note that in a deflection network with the locally optimal routing scheme, round trip propagation delay is unbounded. However, large deviations from the average end-to-end delay between the switches are very rare, so the setup time of accepted sessions should not seriously fluctuate.

The description of the setup phase given above is somewhat simplified. In the protocol implementation, one of the sources would have to initiate the session by sending a "connection request". Then, it would wait for the other party to respond. Only then, would the pair of sources start exchanging dummy packets effectively beginning the phase described above.

This initial stage of the setup phase does not affect the ability of the protocol to provide a long-term quality of service in a deflection network (although it does affect the length of the setup phase). For this reason, in our simulations, the setup phase begins as described above, that is, assuming that both parties already know about each other.

Blocking a Connection

A session is blocked if *any* of the following has happened during the last observation period of F_Q frames:

- The queue of S has grown.

- S receives an SD request from some source and it cannot slow down according to Formula 6.3.

- S receives a connection reject CR from the receiver at $\hat{S}$ (Section 6.2.3).

- S receives a connection terminate CT from the corresponding source $\hat{S}$.

In the first three cases, S will send CT message also to the corresponding sender $\hat{S}$, which must be notified that the session has been blocked.

Accepting a Connection

If during the last observation period of F_Q frames, source S has not been blocked, then it will check if a connection confirm (CC) message has arrived from the receiver. If it has not, S will resume the observation period—a source cannot begin the active phase without making sure that the quality of service perceived by the receiver is fulfilled.

As soon as CC is received, S will send a *connection accept* (CA) message to its corresponding sender $\hat{S}$. This way, $\hat{S}$ will know that as far as S is concerned, the quality of service is likely to be sustained. However, it does not yet mean that the session is accepted—S still does not know if the backward transmission from its corresponding sender $\hat{S}$ may go on without increasing the queue at $\hat{S}$, and if the quality of service perceived by the receiver at S is also fulfilled. In fact, the session may still be blocked. It may happen that $\hat{S}$ is e.g., heavily overflown by packets coming from its neighbours and cannot freely transmit its packets. In that case, $\hat{S}$ will send a connection terminate message (CT) to S blocking the session.

A session may be thus accepted if S receives both CA and CC messages from $\hat{S}$, and $\hat{S}$ receives CA and CC messages from S. From now, the transmission of "regular" packets begins and the session may not be blocked anymore—the active phase begins (see 6.2.1).

After a source sends a CA message to its counterpart, it slightly changes its behaviour. Namely, it can no longer be blocked by other, active sources in the network. This is achieved by changing the setup field in the headers of packets sent from S. Active sources no longer perceive these packets as coming from a setup source. Thus, S behaves like an active source but it may still be blocked by its peer.

6.2.3 The Receiver

A quality of service is described by the maximum packet loss rate P_{max} and the maximum end-to-end delay T_{max}. Similar to the sender, the receiver may be in the setup or active phase. The initial size of the playout buffer, whose purpose is to reassemble possibly misordered packets and smooth out the jitter, is minimal and equals $B = 2$ frames.

A receiver in the setup phase determines if the quality of service it perceives does not exceed $f * P_{max}$ and $f * T_{max}$, where $0 < f < 1$. Fraction f allows the receiver to respond to possible violations of the quality of service before they actually occur.

Every F_Q (the value used also in the transmitter) received frames, the receiver checks if the delay suffered by the received packets is greater than $f * T_{max}$. If it is, the session will have to be blocked—to decrease the delay, the receiver would have to decrease the size of the playout buffer to less than 2 frames. Then, the receiver sends a *connection reject* (CR) message to the sender. The session will be blocked. Before it happens, the receiver will simply refrain from receiving "dummy" data packets arriving from the sender.

If the maximum delay is not exceeded, the receiver checks if the packet loss suffered over the last F_Q frames exceeds $f * P_{max}$. If it does not, the receiver will begin a new observation period of F_Q received frames. After I such periods, if the packet loss and delay have not exceeded their maximum values, the receiver will send a *connection confirm* (CC) message to the source, notifying it that the quality of service may be sustained. The receiver will begin the active phase in which it will only check whether the packet loss does not exceed $f * P_{max}$ (see the explanation below).

If the packet loss exceeds $f * P_{max}$, the receiver checks if the current packet loss is greater than the packet loss recorded previously (i.e., from the previous observation of F_Q frames). If it is, it means that the packet loss increases. In that case, the receiver increases the size of the playout buffer by one frame, and refrains from removing the next frame from the buffer. This will increase the effective size of the playout buffer. Since packets will "stay" longer in the buffer before they are removed and played, fewer packets are likely to arrive too late and be dropped in the future.

If the current packet loss exceeding $f * P_{max}$ is less than the packet loss recorded previously, the receiver will not increase the size of the buffer. As we will explain below, the packet loss may decrease slowly. It means that for some time, it may exceed $f * P_{max}$ despite the fact that no packets are dropped anymore. Obviously, increasing the playout buffer size would be pointless in that case.

In other words, the size of the playout buffer increases only if the current packet loss exceeds $f * P_{max}$ and if it is greater than the previous recorded value of packet loss.

The fact that the receiver may react to a loss of the quality of service before it really occurs is particularly important in the setup phase. Let us assume for a moment that $f = 1$ and that the receiver successfully begins the active phase. Now, if the conditions in the network increase the jitter and some packets arrive to the receiver too late to be played, these packets will be dropped. As we have explained above, the receiver will increase the buffer size—this, however, will also increase the delay. If the delay in the setup phase was very close to T_{max}, T_{max} will be exceeded and this quality of service will be violated. As the simulations show, setting f to 0.5 gives the receiver enough space to accommodate increased packet loss.

High load in the network makes arrivals of packets at the receiver less regular, which in turn increases packet loss. If the loss exceeds $f * P_{max}$, the buffer size grows, increasing the end-to-end delay. At some point, if the load in the network and the packet loss perceived by the receiver are sufficiently high, the size of the playout buffer may increase such that the end-to-end delay exceeds T_{max}.

However, our protocol does not allow the load to overly increase because all sources that inhibit already active sources are blocked. The network load stabilizes at some point so the packet loss, and thereby the end-to-end delay cannot grow beyond T_{max}.

Moreover, the observation time of $I * F_Q$ should allow to detect possible delay violations in the setup phase. Such violations result in blocking the session.

Consider the following scenario.

Assume that 10 packets coming from frame number 5 are dropped at the beginning of a transmission. Assume that so far, only 2500 packets were received (which gives the average of 500 packets per frame). The loss of 10 packets corresponds to 0.004 of all received packets. Even if consecutive frames are received without any loss, the value of packet loss will decrease rather slowly. Packet loss of 0.004 looks bad despite the fact that it actually corresponds to only 10 packets lost in one frame consisting of 500 packets.

It means that the guaranteed packet loss P_{max} may be *temporarily* exceeded. Our protocol, detecting a packet loss exceeding P_{max} will immediately increase the size of the playout buffer. If this buffer appears to be sufficiently large, packets will no longer be dropped. If it does not, the buffer size will further increase until the required packet loss is achieved. This way, the packet loss in the long term will be kept below P_{max}. In other words, we are interested in showing that once a loss violation occurs, the protocol responds appropriately (by increasing its playout buffer) and assures that packet loss will no longer grow.

The minimum setup time after which the receiver decides that the session *should not* be accepted is F_Q frames. After that time, the receiver may realize that the delay in fact exceeds $f * T_{max}$.

The minimum setup time after which the receiver decides that the session *may* be accepted is $I * F_Q$, i.e., after the minimum number of observation periods.

Assuming that the delay threshold is greater than the initial size of the playout buffer (i.e., if $f * T_{max} > 2 * N_F$), the maximum setup time at the receiver is determined by

$$\frac{f * T_{max} - 2 * N_F}{N_F} * I * F_Q$$

where N_F (see Section 6.2.2) is a frame duration. This formula comes from the fact that the receiver starts with the buffer of size 2, and then, every $I * F_Q$, it increases its buffer by N_F up to the maximum at $f * T_{max}$. In the worst case, every increase of the buffer size may take $I * F_Q$ time units because the receiver may find out that the packet loss exceeds $f * T_{max}$ at the end of observation periods.

Clearly, in some cases, this setup time may be long (see Section 6.3). For this reason, both the sender and the receiver could have the ability to send CT (or CR) message to its counterpart if the setup is taking too much time. Such a time-out period would probably depend on the patience of the users. The protocol could automatically send such packets if the setup time exceeded the maximum value. This would indicate some problems in the network or at the other party.

Note that the setup time could be decreased by setting the initial size of the buffer to more than 2 frames. Actually, as we will argue in Section 6.4, this would not be the only benefit of starting the session with a larger buffer.

Note that we do not include jitter as one of the service guarantees. The goal of the playout buffer is to smooth out the jitter so that the traffic perceived by the user will appear to be jitter-free. Large jitter causes packet loss which may in turn force the receiver to adjust the size of its buffer. Thus, there is no need to distinguish jitter as a separate performance measure since implicitly, it is already included in packet loss.

6.3 Results

Synthetic fractal (self-similar) traces used in the simulator were obtained from the traffic generator programmed on the basis of the algorithm given in [RN96]. We have adjusted the load, Hurst parameter and the number M of FRPs so that the synthetic trace obtained from the algorithm is close to the actual trace of a videophone.

The simulation parameters (similar to what we have seen in Section 4.2) are as follows;

- network size is $N = 100$ switches;
- the fraction R of the playout buffer of size B is set to 0.8, that is, the size of the active part of the playout buffer is $B_R = 0.8 * B$;
- packet size is 53 bytes—48 bytes for data and 5 bytes for header (like in ATM networks);
- all links are identical and their lengths are 1 packet (i.e., 424 bits);
- every source in a setup or active phase generates 24 frames per second;
- initial number of active sources in the network is 20;
- every 20 seconds, number $P_c = 4$ (i.e., 8% of the network) of random pairs begin their sessions.

We set the average load λ of the traffic between a pair of active switches to 10 kB per frame. Assuming that frame rate is 24 per second, we obtain the average load of 1.97 Mb/s computed from a sample of 1000 frames.

Hurst parameter of the traffic is uniformly distributed between $H_{min} = 0.60$ and $H_{max} = 0.75$, and the number of FRPs is set to 15;

We have performed simulations for two different connectivities:

- 2-connected network.

  Network capacity is $C = 10$ Mb/s. Assuming that it takes a bit $5 * 10^{-9}$ sec. to travel 1 meter in the medium, having capacity C and a link of 1 packet length, the physical distance between every pair of neighbouring switches is about 8.5 km.

- 4-connected network.

  Network capacity is $C = 5$ Mb/s which, assuming links of 1 packet length, gives the distance of 17 km between the pairs of neighbouring switches.

When a packet arrives at its destination, it is removed from the network and a number of performance measures are computed (see Section 4.1). Note that this time, these measures are computed for every active switch.

We monitor the overall throughput, as well as the packet loss, and end-to-end delay of particular sessions vs the number of activated sources. We consider three environments:

E1 - Without the control mechanism described in Section 6.2. There is no control over the quality of service and the only limit imposed on the number of active pairs in the network is the size of the network.

E2 - With the control variant of the mechanism in which a receiver can neither adjust the size of its playout buffer nor affect the decision about the admission of the new session to the network. This protocol can limit the number of activated sessions but it cannot provide a quality of service. We will see that limiting the number of sessions alone is an efficient way to bound the packet loss. Imposing the specific delay and packet loss requirements, and adjusting playout buffers at the receivers (environment 3) will allow to achieve the required quality of service.

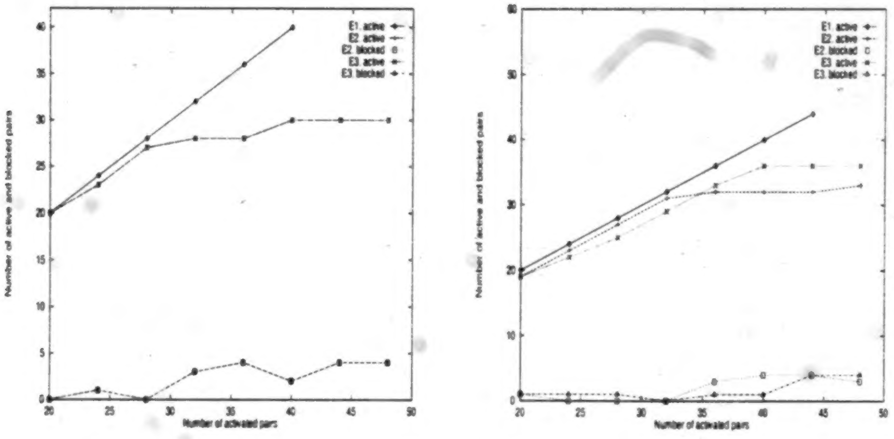
E3 - With the full control mechanism as described in Section 6.2.

In environments $E2$ and $E3$, we also measure how many sessions initiated by new sources are blocked in the setup phase due to the high network load. The initial size of the playout buffer is $B = 2$ frames. Note that this size can change only in environment $E3$. The same pairs attempt to activate their sessions in all three environments.

If we measured the packet loss throughout the simulation by adding all dropped packets and dividing this number by the number of all received packets, its value would be misleading. As we explained in Section 6.2.3, even the loss of several packets could make the value of this performance measure very large for the duration of a whole session. For this reason, packet loss is reset every 20 seconds right before new P_c pairs of sources begin their setup phases. This way, the packet loss shown in the figures below, describes the situation in the network within the given period of 20 seconds. This makes it easier to compare various environments, and it shows how adjusting the playout buffer size helps the receiver sustain the long term quality of service.

Figure 6.2 shows how many new pairs of sources are blocked and admitted to the network with connectivity $k = 2$ and $k = 4$. The number of randomly selected pairs that every 20 seconds try to begin a session is $P_c = 4$. The number of activated pairs in the network grows linearly with the number of pairs that try to begin their sessions only in $E1$. In that environment, every pair of sources that want to begin the session is allowed to do so. The number of blocked sessions, not shown in the figure, is of course 0.

It is not the case in the other two environments. When the number of pairs in the network exceeds some value, new sessions are blocked. That is why, the number of active



(a) k=2

(b) k=4

Figure 6.2: Number of active and blocked pairs vs number of activated pairs

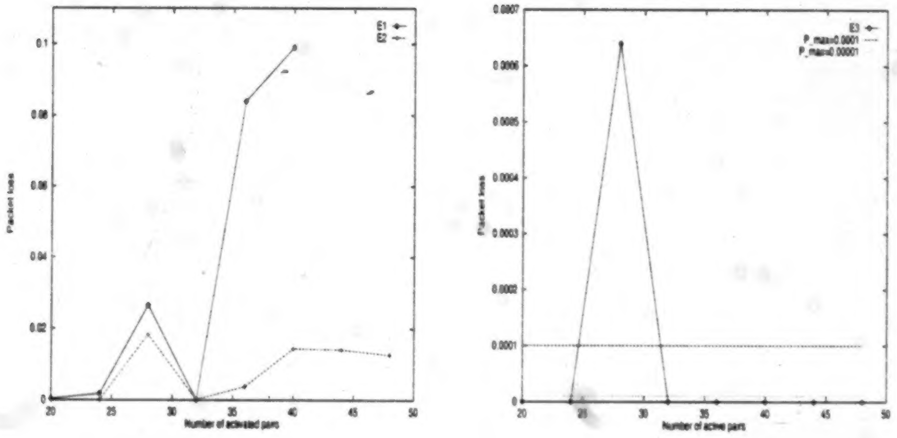
sources eventually becomes almost constant, and the number of blocked sources becomes P_c (which was set to 4). Note that it just happened that the plots for the environments 2 and 3 in the 2-connected network are identical.

We may notice that some sessions are blocked even when the number of pairs in the network is not large (e.g., 28). Apparently, the location of the sources that initiated those sessions was such that the packets were overflowing already active sources. The protocol is able to block these sources. Then, still some pairs of sources may be allowed to activate their sessions as long as their packets do not affect already active sessions.

Figure 6.3 shows the packet loss at one of the receivers that suffered some packet loss almost from the beginning of the session. In the 2-connected network, the receiver was located in the switch number 83 and in the 4-connected network, it was located in the switch number 26. As we have mentioned above, packet loss (and also delay) are reset every 20 seconds so these two performance measures show the situation in the network in those particular intervals. As before, the number of randomly selected pairs that every 20 seconds try to begin new sessions is $P_c = 4$.

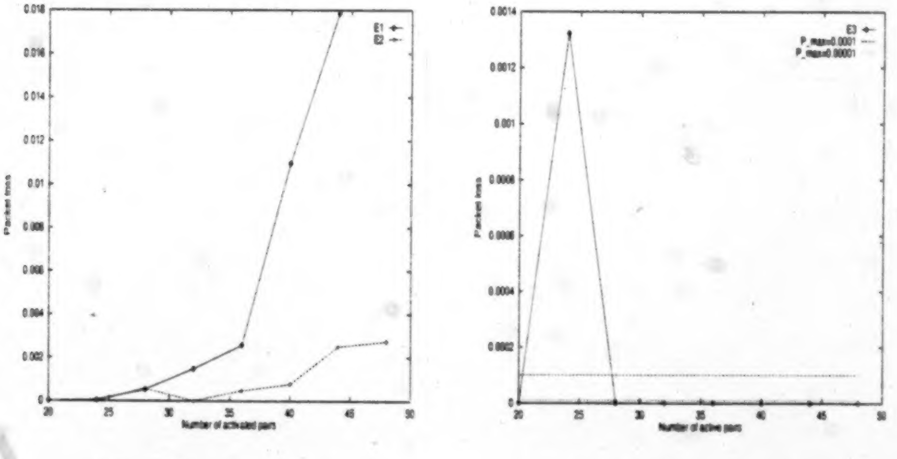
In environment $E1$ (figures 6.3a and 6.3c), packet loss generally increases with the addition of every P_c new pairs. The only exception is for 32 active pairs—apparently, the activity of other sources in the network did not increase the jitter perceived by the receiver to the point in which packets had to be dropped.

Packet loss in environment $E2$ is much lower. It is because of the fact that much fewer



(a) Packet loss, E1, E2; k=2

(b) Packet loss, E3; k=2



(c) Packet loss, E1, E2; k=4

(d) Packet loss, E3; k=4

Figure 6.3: Packet loss vs number of pairs in the 2- and 4-connected network

sessions are admitted to the network. For example, in the 2-connected network, the number of active pairs does not exceed 30 (see Figure 6.2a). Limiting the number of active sources limits the load, jitter and thereby packet loss.

Figures 6.3b and 6.3d show the packet loss at the same receivers as well as the required guarantee imposed on packet loss in environment $E3$. In both cases, i.e., for both values of P_{max}, packet loss was identical.

Indeed, for example in the 2-connected network, after activation of 8 new sessions, so that their total number is 27 (one session is blocked—see Figure 6.2a) the packet loss exceeds P_{max}. In that period of 20 seconds (or 480 frames) 60 packets from one frame were dropped. When the protocol at the receiver side detected this, it increased the playout buffer by 1 frame which was clearly sufficient to sustain zero packet loss in the remaining part of the session.

Similar scenarios were seen in the 4-connected network in which the playout buffer was increased to 3 frames.

Thus, as we have signaled in Section 6.2.3, packet loss may temporarily exceed the allowed maximum value. However, after the receiver adjusts its playout buffer, the required quality of service is achieved.

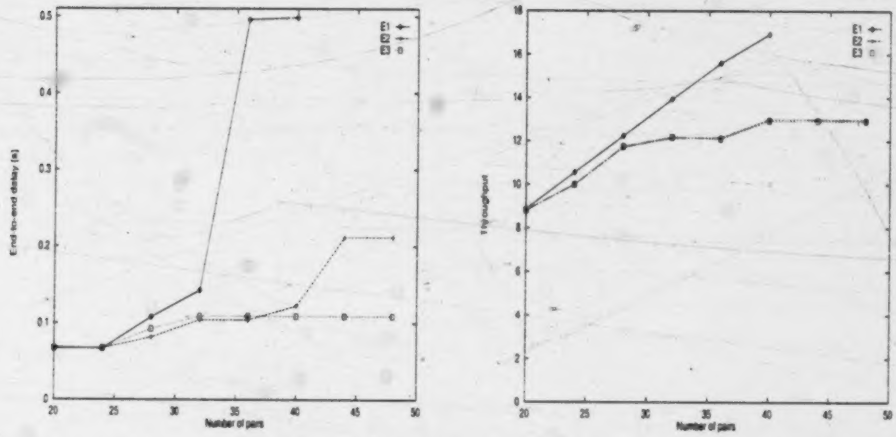
Figure 6.4 shows the remaining performance measures in the 2-connected and 4-connected networks. Again, our attention is focused on receivers at switches 83 and 26, respectively.

The delay perceived at the selected receiver in environment $E1$ increases with the increasing load (figures 6.4a and 6.4c). When the load is high, packets wait longer in the queues before they are inserted to the network. Propagation delay is also increased because a higher number of deflections extend the packets' paths. It all adds up to higher end-to-end delays.

In environment $E2$, in which some sources are blocked, the end-to-end delay is much lower because the load in the network does not grow significantly.

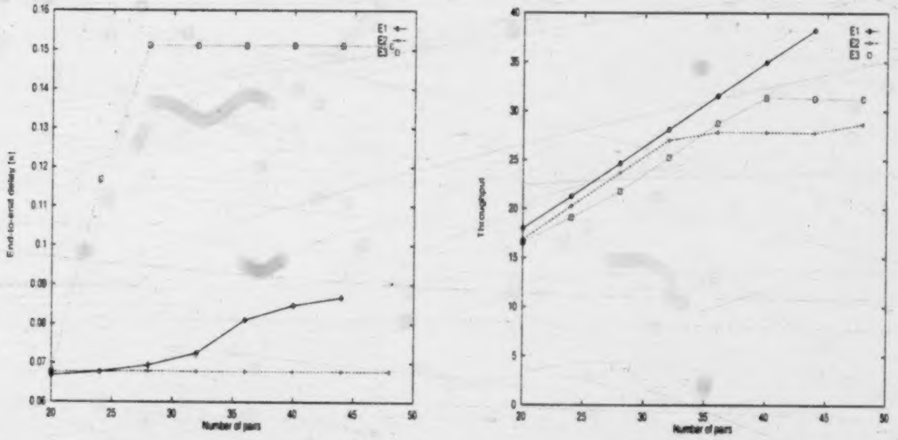
In environment $E3$ the delay is most regular. It increases sharply at the beginning particularly in the 4-connected network. This corresponds to the moment at which the receiver's buffer was increased by 1 frame in the 2-connected network and 2 frames in the 4-connected network. From that moment, packets arriving to the receiver must have spent more time in the buffer before they were played. Notably, the delay in the remaining part of the simulated session was almost constant.

The throughput in environment $E1$ increases regardless of the network load. This is one



(a) End-to-end delay, k=2

(b) Throughput, k=2



(c) End-to-end delay, k=4

(d) Throughput, k=4

Figure 6.4: End-to-end delay and throughput vs number of pairs in the 2- and 4-connected network in all three environments

more feature that makes deflection networks attractive. We may notice, however, that the increase is slightly smaller for a greater number of activated pairs. It is caused mainly by the packet loss that decreases the number of successfully received packets, thereby decreasing the throughput. As we could expect, in two other environments the throughput is smaller (by about 20% for the highest load) which is caused by the fact that there are fewer active sources in those networks. Note that the shapes of the plots showing the throughput and the number of active pairs (Figure 6.2) are similar.

The length of the setup time in environment $E2$ is slightly above $I * F_Q = 20$ frames (recall that 24 frames correspond to 1 second). After the observation period of $I * F_Q$ frames, the source will know if its queue grows and if it disturbs other sources. It will thus either send connection terminate CT or connection accept CA to the corresponding source. Similar messages received from the corresponding source complete the setup phase.

The setup time is longer in environment $E3$, even if the receiver does not have to increase its playout buffer. It is because the sender cannot send a CA message to the corresponding sender before the arrival of a connection confirm CC message from the receiver. The CC message, on the other hand, cannot arrive to the sender during its first observation period of $I * F_Q$ frames because the observation period at the receiver has the same length. It also begins after first setup packets are received.

In fact, in the implementation of this protocol, the setup time would be even somewhat longer. Recall that we have not considered the time needed by a source to "call" another source.

In our simulations, the average setup time observed in environment $E2$ was about 1 second, and the setup time in environment $E3$ was about twice as long.

6.4 Possible Improvements of the Protocol

In the setup phase, a new source "probes" the network transmitting dummy frames. If there is enough bandwidth available in the network, the pair of sources will be allowed to begin their actual session. To fulfill its purpose, the dummy frames should well represent the session. It may make sense to keep *signatures* of typical sessions, i.e., pre-generated sequences of frames to be used for session probing during the setup phase. Such signatures could be viewed as implicit QoS specifications. A shaping/policing scheme at the source would make sure that the outgoing traffic conforms to the specification (burstiness, peak and average rate) implicitly encoded in its signature. This way, we could be certain that if a new source does not disturb already active sessions in the setup phase, it will not impair their performance later. Slight increases of the traffic caused by the new sources transmitting packets in their setup phases, could still be absorbed by receivers.

We mentioned in Section 6.2.3 that the receiver could start its setup phase with a buffer size greater than 2. This would not only give a shorter setup time but could also assure that no packet will be dropped during the session. Note that for example in our simulations (see Figure 6.3b), buffer of size 3 would give zero packet loss throughout the session. Estimation of the initial size of the buffer B is not difficult. The receiver should pick maximum B such that:

$$B * N_F + T_p < f * T_{max}$$

holds, where T_p is the access and propagation delay, and N_F is the duration of the frame. This would still give the receiver some space to increase its playout buffer (note the f in the formula) if the packet loss somehow increased beyond $f * P_{max}$.

A precise value of T_p is not known—it depends on the propagation delay and the time the packets spend in the sender's queue. Both these values may vary. However, recall that the queues at the transmitters are not likely to grow without a bound because new sessions that could cause that are blocked. Thanks to this, the load and thereby the number of hops (that determines the propagation delay) also remain manageable.

The receiver could approximate the value of T_p by checking the moments at which packets passing through its switch were generated.

It is easy to simplify this protocol if its complexity appears to be too high. To do it, a setup source that receives a slow-down message SD *must* be blocked—it cannot slow-down. This way, the setup phase will become shorter, and active sources will not have to distinguish between packets sent from the setup source via different outgoing links. The header format will no longer include fields carrying the link number through which the packet was transmitted and the transmission rate concerning this link.

The only downside is the decreased throughput. Since every SD message received by a setup source forces it to block the session, fewer pairs will be allowed to initiate their sessions. However, our simulations show that the throughput decrease is very low, about 3%.

6.5 Summary

Our results indicate that limiting the number of active sources coupled with the receivers' ability to adjust the size of their playout buffers allows the network to provide a long term quality of service. Notably, no resources are reserved in advance and no bandwidth is wasted on tokens. This also suggests that this protocol is easily scalable. In fact, with the increasing size of the network and/or network's capacity, only the setup phase need to be extended. In that phase, sources initiating a session must collect confirmations from each other.

The results obtained from simulations of triangle networks (not shown here) are similar. That is, the protocol seems to work properly even in networks with irregular topologies.

It seems that other applications would require some minor changes from the protocol. For example, in the setup phase of a videoconference, the connection accept (CA) message would be multicast. Every source has to wait for CA from all other sources before beginning the active phase. The arrival of at least one connection terminate CT message, which would often be multicast as well, would block the session (or perhaps trigger a new attempt with reduced QoS requirements).

Transmission of video movies would not require CA messages—a video source has no corresponding sources. The source would only make certain that its transmission is possible without disturbing active sessions and that the quality of service is provided as required by the receivers.

Let us review the properties of this protocol against the list of properties expected from the ideal protocol (see [DG93]).

1. Simplicity.

   This protocol is simple. The number of control messages required to establish a connection is also very low. As we have mentioned, no resources are reserved in advance which also simplifies this protocol.

2. Fairness.

   This protocol does not provide perfect fairness. It may still happen that switch S wishing to initiate its session is overflown by packets transmitted by highly active neighbours even if the rest of the network is relatively idle. However, if those active neighbours are in the setup phase, S may force them to slow down or even block their sessions.

3. Throughput in the network should not decrease with the increasing transmission rate or network size.

   True. In fact, the throughput in deflection network, understood as the ratio of received bits to time, increases with the increasing size of the network. It must be pointed out, however, that the setup phase will also increase under such conditions. The maximum round-trip delay, which affects the observation time, increases together with the increasing size of the network.

   Note that a higher response time and, consequently, shorter setup phase, may be achieved by giving control packets priorities over data packets.

4. Average access delay should approach 0 as the load approaches 0.

   True. When the network load is very low, the sources simply insert their packets in all free slots they perceive.

5. The protocol should accommodate heterogeneous traffic demands, guaranteeing a finite maximum packet delay for synchronous traffic and a sustainable throughput for asynchronous traffic.

   A finite (although unbounded) maximum packet delay is assured in this protocol. Large deviations fron the average interarrival delay are extremely rare regardless of the network load and traffic.

   If the sources of asynchronous traffic are treated in the same way as the sources of synchronous traffic, throughput of the asynchronous traffic should not suffer. However, as we saw in Chapter 5, it may make sense to give the "more important", synchronous traffic priority over the asynchronous traffic. In this case, packets originating at synchronous sources may throttle asynchronous sources decreasing their throughput.

6. The protocol should carry synchronous traffic of variable intensity, up to using the whole bandwidth of the network.

   The protocol can certainly carry traffic of variable intensity but, in contrast to the uncontrolled network (environment $E1$), the maximum throughput is slightly lower. There is an obvious trade-off between the throughput and quality of service.

7. The protocol should be self-synchronizing, so that jitter remains negligible.

   Jitter is not negligible in deflection networks. Packets may follow different paths and arrive at destinations misordered. This certainly increases the jitter in the network. However, as we have pointed out in Chapter 2, jitter does not seem to be seriously affected by the traffic patterns in the network and transmission rates of the sources. Thus, the reassembly buffer of a reasonable size can smooth out the traffic arriving at the destination. Moreover, the destination may dynamically change the size of the buffer reacting to a possible increase of jitter.

8. The protocol should be predictable, so that a critical failure can be recognized by at least one station in time not exceeding maximal end-to-end delay.

   True. The ways of detecting and handling failures in deflection networks are described in [Max87].

Chapter 7

Asynchronous Deflection with Transient Buffers

In this chapter (see also [OG98a]), we investigate several routing algorithms for asynchronous deflection networks, i.e., ones that operate in an unslotted manner. We determine the impact of an extra input buffer space on the quality of routing decisions. Finally, we compare the performance of asynchronous and synchronous deflection networks.

Traditionally, deflection networks have been viewed as slotted systems. In such a network, packets arrive at a switch in orchestrated batches and are examined simultaneously. This way, a routing decision deals with all packets that arrive at the switch within the current *slot* and accounts for the combined preferences of all these packets [Max85, Max87, Max89, Max91, MK93]. This approach poses some implementation problems: in a realistic environment, transmission rates of different switches may not be exactly the same, and the synchronization of the network may be difficult. To keep the slot arrival rate steady, a switch may need additional buffer space and/or the network may have to resort to complicated backpressure mechanisms [Max88]. As the performance of such techniques depends on the (normalized) propagation delays across the network, they do not scale very well to the increasing network size and/or transmission rate. The fixed packet (slot) size is another drawback.

Alternatively, one may consider asynchronous networks, in which packets (not necessarily of the same length) may arrive at a switch spontaneously [GM93, MWW90a, MWW90b]. With the simplest implementation of this idea, the switch will make a routing decision for one packet at a time, as soon as the packet's header (destination) has been recognized. Asynchronous routing decreases the complexity of synchronous networks but it tends to make significantly worse routing decisions, particularly at low connectivities (e.g., 2×2 switches). On the other hand, as pointed out in [GM93], the quality of asynchronous routing decisions improves with the increasing connectivity of the switch.

To see why in some cases asynchronous deflection routing may perform significantly worse than synchronous routing, consider the following scenario at a 2×2 switch. Packet P_1 arrives at the switch and both output ports are idle. Assume that the packet doesn't clearly prefer any output port, i.e., the switch is free to choose any of them to relay the packet. Suppose the switch chooses port p_1. A moment later, while P_1 is still being relayed, another packet, P_2, arrives at the same switch. This time P_2 prefers p_1, i.e., the port that is being occupied by P_1. There is no choice but to deflect the new packet, although if both packets were available at the same time, the switch would be able to avoid the deflection. Note that if another packet, say P_3, arrives while P_2 is being deflected, the switch will have absolutely no flexibility as to the fate of that packet. Consequently, it may get into a sustained scenario of unbounded duration in which all packets arriving at the switch are continuously being deflected. This phenomenon is particularly harmful in networks with small connectivities and under heavy load.

The above problem can be alleviated by equipping the switch with some transient buffer space. This space will make it possible to postpone a routing decision for a while, until it can be confronted with the routing decisions for other packets. In this chapter we discuss a few possible implementations of this idea and their impact on the performance of asynchronous deflection routing.

7.1 Network Model

We consider the torus topology (Figure 1(a)), which is the standard configuration for Manhattan-street networks (MSN) [Max85]. Notably, other topologies that we tried (including some biased topologies, e.g., a triangle) produced performance results highly consistent with those obtained for the torus. In relative terms, the results presented in this chapter should hold for many other reasonable topologies.

Every switch is equipped with some buffer space; the way of using these buffers is part of the routing strategy. The network operates in an asynchronous manner, in a way described in the Introduction. Packets are not explicitly aligned at the switch, they may be of different length and they are allowed to arrive at any time.

Every switch has a *host* capable of contributing traffic to the network. The traffic is uniform which means that all sources and destinations are equally probable. Every host generates packets according to the Poisson distribution with a given mean. The load parameter determines how many new packets appear in the entire network during the time needed to transmit a single packet. To reduce the number of parameters and eliminate irrelevant degrees of freedom, we keep the packet length fixed, although the routing rules never pose this requirement. Our experiments carried out for variable packet length (see Section 7.2.3) have produced similar results to those where the packet length was fixed. The actual packet length is immaterial: what matters is the ratio of the buffer size at a switch to the (average) packet length.

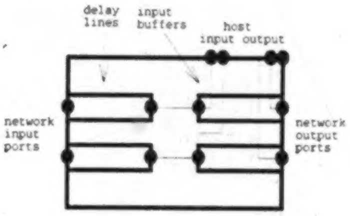


Figure 7.1: Simplified model of a switch

To avoid disrupting the relayed traffic by its own packets, every switch employs delay lines separating its input ports from the routing fabric (including the buffer space). If a switch whose host is backlogged finds at least one delay line free (see Figure 7.1), it removes the first outgoing packet from the host's queue and inserts it into the input buffer—as if the packet arrived from the network on the corresponding input port. With the delay line, the switch makes sure that the packet transmitted by the switch will not collide with a packet entering the buffer from the input port. The minimum size of a delay line must be L, where L is the maximum length of a packet.[1] With this approach, the switch is never forced to drop a packet, which we view as a fundamental property of deflection routing. If an incoming packet is addressed to the current switch (i.e., its host), the switch receives the packet and removes it from the network.

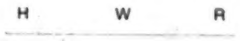


Figure 7.2: Input buffer

The input buffer consists of three logical parts corresponding to three stages of packet processing—as shown in Figure 7.2.

- Part H absorbs the packet for the amount of time needed by the switch to recognize the packet's header, more specifically, the packet's destination. A routing decision regarding the packet can only be made if its destination is known.

---

[1] An internal switch, e.g., one that is not connected to a host, need not be equipped with delay lines.

- Part W is the "waiting room." It allows the switch to defer the routing decision for some time, e.g., to coordinate it with routing decisions for other packets. This part may be empty, in which case the routing decision for the packet must be made as soon as the first bit of the packet has passed the H part.
- Part R corresponds to the amount of time needed by the switch to actually make a routing decision.

In the sequel, H, W, and R will be used to denote the amount of time spent by a packet in the three components of the input buffer. As is customary in homogeneous networks, we will use one bit as a time unit. We assume that H and R are fixed, but W can be varied. If $W = 0$, the total length of the input buffer is the shortest possible. Note that the delay lines used by the hosts are in fact separate buffers.

A routing decision made at time t is performed $H + W$ bits after the first bit of the first packet affected by this decision entered the input buffer. Let us denote this packet by P. If $W \neq 0$, the routing decision may involve packets that arrived later than P, assuming that their destinations have been recognized (the packets have passed through H) before time t. We say that such a routing decision is *initiated* by packet P.

Our primary performance criterion is the maximum throughput achievable by the network expressed as the total number of received bits per one bit of time. A single simulation experiment was run as long as the differences among the four consecutive snapshot values of throughput taken at intervals corresponding to 50 times the maximum propagation distance in the network were more than 1% (i.e., until the throughput reached the equilibrium).

For the results presented in this chapter, the simulation parameters were as follows:

- network size $N = 100$ switches
- packet length $L = 1024$ bits[2]
- $R = H = 102$ bits (i.e., 1/10 of L)
- link length $l = 4000$ bits (almost 4 packets)
- delay line length $d \geq L$ (this was one of the variables)

---

[2] In Section 7.2.3 we present some results for variable packet length.

Note that R and H are practically irrelevant—they just slightly inflate the propagation distance between a pair of neighbouring switches. As it turns out (Section 7.2.2), sometimes it makes sense to increase the length of the delay lines (d) above the minimum (L); thus, this parameter was varied in some experiments. Retaining the torus topology, we have investigated networks with connectivities $k = 2, 4, 8$. If $k = 4$, all links in Figure 1(a) are bidirectional; with $k = 8$, four diagonal bidirectional links are added to every switch. The most important variable was W, i.e., the "useful" length of the input buffer.

7.2 The Standard Algorithm

The most natural routing algorithm routes the packets arriving on the same input port in the order of their arrivals. Thus, a packet can only be routed if no other packet precedes it in the input buffer. If $W > 0$, it is possible that more than one packet will compete for the same free outgoing port. In such a case, the *locally optimal* routing decision [BC90, GM93] assigns free output ports to all competing packets in such a way that the sum of the shortest distances to their destinations is minimized. If several possible selections give the same minimum sum, one of them is chosen at random. This randomization was postulated in [Max91] as a way of eliminating *livelocks* in MSN. Note that in asynchronous deflection networks, which are inherently nondeterministic, the problem of livelocks is less serious.

The routing algorithm operating according to the above rules will be called *standard*.

7.2.1 Performance of the Standard Algorithm

	k=2		k=4		k=8	
W	max.	saturated	max.	saturated	max.	saturated
0	5.530	3.495	14.288	12.897	70.726	70.416
128	5.597	4.143	17.829	14.950	72.957	72.757
256	5.837	4.459	18.605	15.456	69.204	68.340
512	6.086	5.135	18.322	14.605	60.775	56.018
1024	7.212	5.543	15.156	11.825	46.231	36.231
1536	6.471	5.349	13.545	10.276	38.195	30.589
2048	5.687	4.991	12.004	9.682	37.507	26.504

Table 7.1: Maximum and saturated throughput for different W and k (standard algorithm)

Table 7.1 shows the maximum achievable throughput and the saturated throughput[3] in the asynchronous torus network for several waiting times W and different connectivities.

---

[3] The reason why the saturated throughput in an asynchronous deflection network tends to drop is explained in [GM93].

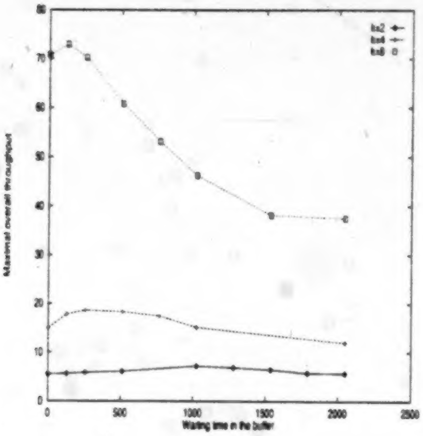


Figure 7.3: Maximum overall throughput vs waiting time W in the buffer for different connectivities

Figure 7.3 shows the maximum throughput achieved for a given connectivity and W.

One would expect that the maximum throughput will increase with increasing W, because the quality of a routing decision will tend to improve. However, this only holds for some ranges of W. In particular, if $k = 2$, the throughput for $W = 1024$ is higher than for $W = 0$, but it drops for larger W. If $k = 4$, the throughput reaches its maximum for $W = 256$ and then drops. If $k = 8$, the throughput reaches its peak for an even smaller value of $W = 128$.

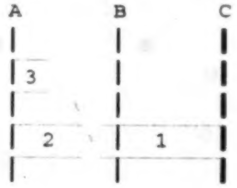


Figure 7.4: Standard algorithm, $k = 2$

Let us start with the 2-connected network. To understand why the throughput drops when $W > L$, consider the scenario illustrated in Figure 7.4. Lines A, B, and C show different left (input) boundaries of the buffer, corresponding to different values of W. C which is also the right boundary of the buffer, corresponds to $W = 0$. Without missing anything important, we can assume that $R = H = 0$.

Consider case A. Packet 1 arrives at the end of the buffer (C) first and competes with packet 3 (it does not compete with 2 because 2 is behind 1 in the same buffer). Thus, we have two competing packets and neither of the output ports is busy (notation $C(2,0)$). After a while, packet 2 reaches C. There is no packet for it to compete with (the fate of 3 has already been determined), and there is only one available output port (the one freed by packet 1). We denote this situation by $C(1,1)$. The history of the routing decisions is $C(2,0), C(1,1)$.

In case B, when packet 1 reaches C, packet 3 is not yet available; thus, the first routing decision is $C(1,0)$. When packet 2 gets to the end of the buffer, it will compete with 3 ($C(2,0)$). The two routing decisions are $C(1,0), C(2,0)$.

When $W = 0$ (case C), all three packets will be routed independently: $C(1,0)$, $C(1,0)$, $C(1,1)$.

The best chance for all packets to be routed over their preferred links is in case B. The first packet gets its prefered port and at least one (but possibly both) of the remaining two packets gets its prefered port as well.

Next is case C. The reason why it is worse than B is that the last packet (3) must be routed via the only available port, so its chances for getting the prefered port are smaller than in the previous case.

Case A (the largest buffer) turns out to be the worst of them all! First we have a competition ($C(2,0)$) in which one packet gets its prefered port for sure, but the other one may be deflected. This is followed by $C(1,1)$—a no-choice relay—in which packet 3 may also suffer a deflection.

We can see that the best routing decisions in the above scenario are made when W is greater than 0 but smaller than L. Table 7.2 shows the frequencies of the particular types of routing decisions under heavy load for different values of W. They were obtained by observing routing at every switch and counting how many times a switch makes a particular decision (i.e., C(1,0), C(2,0) or C(1,1)).

W	C(1,0)	C(2,0)	C(1,1)
0	0.200	0.003	0.797
1024	0.046	0.224	0.730
1536	0.018	0.225	0.757
2048	0.014	0.226	0.760

Table 7.2: Standard algorithm, load 14, k=2

When $W = 0$, the percentage of potentially bad routing decisions $C(1,1)$ is higher than for $W = 1024$. Then as W increases, the frequency of $C(1,1)$ increases as well. At the same time, the percentage of good decisions $C(1,0)$ decreases.

When the input buffer is longer than L, it is more likely that a packet P_2 following packet P_1 that initiates a routing decision will be ahead of another packet P_3 competing with P_1. In this scenario, P_2 will have only one link to be routed over. If buffers are smaller, this situation is less likely to happen.

The same phenomenon is observed for connectivities higher than 2. Note that the throughput for $k = 4$ and $k = 8$ starts to drop before W reaches L (Table 7.1 and Figure 7.3). The reason for this is given below.

One observation that one can make is that a smaller W causes splitting a large routing decision into several smaller decisions involving fewer packets. In the extreme case of $W = 0$, every routing decision concerns only one packet (unless two packets arrive at the switch at exactly the same time). Then the sustained deflection scenario mentioned in the introduction is likely to occur, especially if $k = 2$. But even in a 4-connected network, if four packets are routed to the output ports and then another packet arrives and reaches the end of the input buffer, it may not have too many output ports to select from. In particular, if it follows the packet that initiated the routing decision, there will be only one port available. If the buffers are not very long, fewer packets (e.g., 2) may be routed at a time, so packets following them will be more likely to find more output ports available. But this can only happen if consecutive packets arriving at the switch over the same link are separated by some gaps. If every packet is immediately followed by another, W becomes irrelevant because then (ignoring the extremely rare cases of two or more packets arriving at exactly the same time) every packet is routed individually. This also explains why the throughput in a saturated network tends to drop: the benefits of a nonzero W tend to disappear when packets become more closely spaced.

To understand the reason why the saturated throughput for $k = 4$ and $W = L$ is even worse than for $W = 0$, consider the scenario shown in Figure 7.5.

In case A, W is large but still somewhat smaller than L. First, packet 2 competes with packet 1 in $C(2,2)$. Note that it cannot compete with with 3 or 4 because these packets are preceded by packets that are being routed. Then, the routing decisions for packets 3 and 4 are both $C(1,3)$. The history is $C(2,2)$, $C(1,3)$, $C(1,3)$.

In case B, packet 2 is routed alone in $C(1,2)$. Then, 1 competes with 3 in $C(2,2)$. Finally, packet 4 finds three outgoing ports busy, so the routing decision is $C(1,3)$. The history is $C(1,2)$, $C(2,2)$, $C(1,3)$.

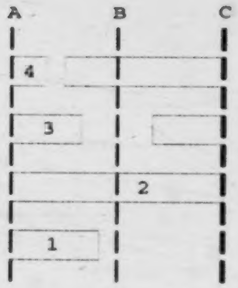


Figure 7.5: Standard algorithm, $k = 4$

With $W = 0$, packet 2 is routed as before ($C(1,2)$). Then, packet 1 is routed alone in $C(1,2)$ (the predecessor of packet 3 has left the buffer, so again two output ports are available). Finally, packets 3 and 4 reach C and each of them finds three ports busy. The history is $C(1,2), C(1,2), C(1,3), C(1,3)$.

In the last case, we have two no-choice relays $C(1,3)$. In the second case, the situation is better: there is only one no-choice routing, one "reasonable" routing ($C(1,2)$) and one "passable" decision ($C(2,2)$). In case A, we have two no-choice relays $C(1,3)$ and one "passable" decision $C(2,2)$. The routing history in this case (with the largest buffer) is surprisingly the worst!

W	C(1,0)	C(2,0)	C(3,0)	C(4,0)	C(1,1)	C(2,1)	C(3,1)	C(1,2)	C(2,2)	C(1,3)
0	0.038	0.000	0.000	0.000	0.155	0.001	0.000	0.391	0.004	0.411
256	0.020	0.014	0.007	0.002	0.063	0.075	0.032	0.218	0.170	0.399
512	0.018	0.018	0.014	0.008	0.032	0.067	0.064	0.134	0.211	0.434
1024	0.006	0.007	0.009	0.014	0.003	0.019	0.065	0.048	0.249	0.581
2048	0.003	0.004	0.004	0.011	0.002	0.006	0.061	0.009	0.259	0.642

Table 7.3: Standard algorithm, load 32, k=4

Table 7.3 shows the frequencies of the particular types of routing decisions for $k = 4$. When $W = 0$, the percentage of potentially bad routing decisions $C(1,3)$ is higher than for $W = 256$. Then, with increasing W, the frequency of $C(1,3)$ increases, and for $W = 512$ it is even greater than for $W = 0$. We may also notice that the percentage of good decisions $C(1,0)$ or $C(1,1)$ decreases.

Due to the large number of possible routing decisions, we do not show their frequencies for the 8-connected network.

7.2.2 Standard Algorithm with Longer Delay Lines

In the previous section, we noticed that small gaps between consecutive packets arriving on the same link tend to worsen the quality of routing decisions. One natural way to increase these gaps is to increase d—the length of the delay lines—beyond L. As we increase packet spacing, the maximum throughput achievable by the network will tend to decrease because of the incurred bandwidth wastage. However, the improved quality of routing decisions may compensate for this loss and outweigh it. Clearly, there must be a middle ground somewhere.

	k=2, W=1024		k=4, W=256		k=8, W=128	
F	max.	saturated	max.	saturated	max.	saturated
0	7.212	5.543	18.605	15.456	72.957	72.757
1.0	7.948	7.963	20.705	20.491	77.935	77.933
2.0	8.157	8.049	21.286	20.721	72.870	72.260
3.0	7.923	7.892	20.762	19.980	66.415	65.159

Table 7.4: Maximum and saturated throughput for different F, W, and k (standard algorithm with longer delays)

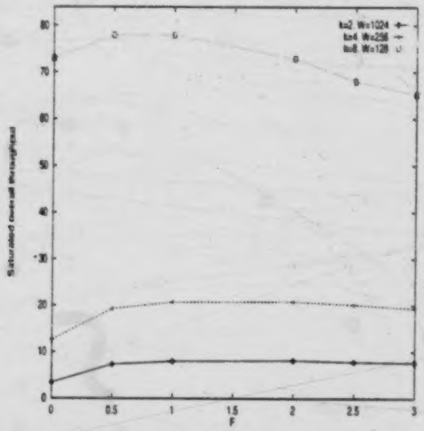


Figure 7.6: Saturated throughput vs F for different connectivities

Let F be the fraction by which the length of the delay line d is increased above L. For example, if $F = 0.5$, $d = L + 0.5 * L$. Table 7.4 and Figure 7.6 show the maximum and saturated throughput for different connectivities k and for the buffer size W for which the observed throughput (see Figure 7.3) was maximum. This table suggests that increasing d up to $3L$ ($F = 2$) in a network with $k < 8$, and $2L$ ($F = 1$) for $k = 8$ results in some

F	C(1,0)	C(2,0)	C(1,1)
0.0	0.201	0.003	0.796
1.0	0.181	0.386	0.433
2.0	0.323	0.360	0.317
3.0	0.415	0.328	0.257

Table 7.5: Standard algorithm with longer delay lines, load 14, W=1024, k=2

F	C(1,0)	C(2,0)	C(3,0)	C(4,0)	C(1,1)	C(2,1)	C(3,1)	C(1,2)	C(2,2)	C(1,3)
0.0	0.038	0.000	0.000	0.000	0.156	0.001	0.000	0.391	0.003	0.411
1.0	0.064	0.046	0.014	0.002	0.191	0.117	0.020	0.284	0.086	0.176
2.0	0.128	0.059	0.014	0.001	0.236	0.098	0.014	0.247	0.062	0.141
3.0	0.176	0.066	0.012	0.001	0.260	0.091	0.011	0.227	0.049	0.107

Table 7.6: Standard algorithm with longer delay lines, load 32, W=256, k=4

improvement in the achievable throughput, with the most significant improvement observed for $k = 8$ and $F = 1$. Tables 7.5 and 7.6 show the distribution of the routing decisions for different values of F.

7.2.3 Variable Packet Length

In this section, we show the performance results for the standard algorithm with variable packet length. For these experiments, the minimum and maximum packet length was set to 128 and 2048 bits, respectively. The actual length of every packet was determined by Poisson distribution with the mean of 1024 bits. Packets shorter than 128 bits were inflated to the minimum legal size, packets longer than 2048 bits were split into several smaller packets. The remaining parameters were left as before, except for d which had to be increased to 2048 bits.

	k=2		k=4		k=8	
W	max.	saturated	max.	saturated	max.	saturated
0	5.548	5.271	19.103	18.550	75.765	75.571
128	5.597	5.706	20.157	20.045	77.890	77.354
256	6.222	6.092	20.796	20.546	75.756	75.059
512	6.959	6.733	20.212	20.024	68.702	68.561
1024	7.914	7.898	17.299	16.753	51.451	50.958
1536	7.057	7.035	14.063	13.362	39.186	36.307
2048	6.500	6.394	12.074	11.682	38.279	31.313

Table 7.7: Maximum and saturated throughput for different W and k (variable packet length)

W	C(1,0)	C(2,0)	C(1,1)
0	0.387	0.002	0.611
1024	0.175	0.390	0.435
1536	0.082	0.425	0.493
2048	0.039	0.414	0.547

Table 7.8: Standard algorithm (variable packet length), load 14, k=2

W	C(1,0)	C(2,0)	C(3,0)	C(4,0)	C(1,1)	C(2,1)	C(3,1)	C(1,2)	C(2,2)	C(1,3)
0	0.087	0.000	0.000	0.000	0.291	0.001	0.000	0.403	0.001	0.217
256	0.065	0.047	0.013	0.002	0.193	0.113	0.019	0.287	0.086	0.175
512	0.037	0.058	0.040	0.010	0.110	0.137	0.051	0.222	0.118	0.217
1024	0.011	0.032	0.063	0.051	0.021	0.063	0.059	0.164	0.168	0.368
2048	0.003	0.004	0.013	0.049	0.004	0.021	0.101	0.042	0.237	0.526

Table 7.9: Standard algorithm (variable packet length), load 32, k=4

Table 7.7 shows the maximum and saturated throughput in the 2, 4 and 8-connected networks. Tables 7.8 and 7.9 show the type frequencies of routing decisions for several values of W and connectivities 2 and 4. Relations between the throughput and the waiting time are similar to what we have seen in the previous sections.

The results for the variable packet length tend to be better than those presented in Section 7.2.1 (especially for higher connectivities), because we get here (without asking for it) the benefits of increased d (discussed in the previous section). Note that the delay line must accommodate the longest packet; therefore, if many packets are shorter than the maximum, they will tend to be separated by gaps without any additional measures.

7.3 The "Quick" Algorithm

One possible way to improve the performance of the "standard" algorithm is to speed up routing in those cases when a packet's fate becomes known before the packet has reached the end of the input buffer. Such a packet will be immediately forwarded to the selected output port, without having to go through the entire buffer. Besides decreasing the amount of time spent by the packet in the buffer, this approach will also improve the chance that the next packet will find more output ports free. One drawback of this solution is a more complicated organization of the buffer space and, consequently, more expensive switch hardware.

k=2			
W	algorithm	max.	saturated
1024	S	7.212	5.543
512	Q	15.108	14.726
1024	Q	16.901	16.825
1536	Q	16.643	15.587
2048	Q	15.598	14.999
k=4			
256	S	18.605	15.456
256	Q	26.993	25.983
1024	Q	42.797	42.193
2048	Q	32.797	32.794
k=8			
128	S	72.957	72.757
128	Q	84.766	84.426
512	Q	132.436	132.124
1024	Q	83.006	82.804

Table 7.10: Maximum and saturated throughput of standard (S) and quick (Q) algorithms

7.3.1 Comparison with the Standard Algorithm

Table 7.10 compares the maximum and saturated throughput achieved by the standard and quick algorithms. The delay lines were L bits long in both cases.

The throughput achieved by the quick algorithm is significantly higher than that of the standard algorithm. For $k = 2$ and 4, the improvement factor exceeds 2, and it comes close to 2 for $k = 8$. Tables 7.11 and 7.12 show the frequency of particular classes of routing decisions for $k = 2$ and 4.

W	C(1,0)	C(2,0)	C(1,1)
512	0.187	0.616	0.197
1024	0.214	0.783	0.003
1536	0.183	0.785	0.032
2048	0.192	0.723	0.085

Table 7.11: Quick algorithm, load 20, k=2

It is interesting to note that the throughput increases in a wider range of the buffer size W. For example, if $k = 4$, the throughput for $W = 1024$ is higher than the throughput for $W = 256$ (which is the optimum buffer size for the standard algorithm). This is because many packets do not have to go through the entire buffer and, from their perspective, the buffer appears shorter than it is.

W	C(1,0)	C(2,0)	C(3,0)	C(4,0)	C(1,1)	C(2,1)	C(3,1)	C(1,2)	C(2,2)	C(1,3)
256	0.018	0.034	0.042	0.023	0.080	0.130	0.082	0.198	0.168	0.225
1024	0.013	0.065	0.205	0.271	0.025	0.067	0.078	0.071	0.064	0.141
2048	0.022	0.074	0.154	0.151	0.049	0.094	0.081	0.118	0.092	0.165

Table 7.12: Quick algorithm, load 50, k=4

7.3.2 Longer Delay Lines

A natural next step is to see how the quick algorithm will perform when we increase *d*—the length of the delay lines—which enforces packet spacing. This idea proved beneficial for the standard algorithm (Section 7.2.2).

	k=2, W=1024		k=4, W=256		k=8, W=128	
F	max.	saturated	max.	saturated	max.	saturated
0	16.901	16.825	26.993	25.983	84.346	83.305
0.5	13.891	13.478	26.368	25.443	83.858	82.954
1.0	13.377	12.604	24.737	24.674	80.668	80.565
1.5	11.462	11.374	24.351	24.192	78.326	78.129
2.0	10.879	10.830	24.106	23.757	74.123	73.844

Table 7.13: Maximum and saturated throughput for different F, W, and k (quick algorithm with longer delay lines)

The results of this experiment are shown in Table 7.13. Surprisingly, increasing d does not help at all. The best results were consistently observed for $F = 0.0$, i.e., $d = L$. To understand why, consider the following example illustrated in Figure 7.7.

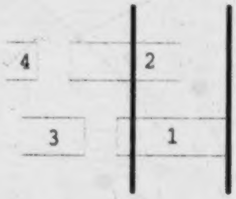


Figure 7.7: Quick algorithm with longer delay lines, $k = 2$

The input buffer is visualized as the space between the two vertical lines. The presented scenario is quite frequent in a heavily loaded network. Packet 1 initiates a routing decision and competes with packet 2 ($C(2,0)$). When packet 3 reaches the end of the buffer, it will find both outgoing ports free—packet 2 will have been removed by then because it does not have to pass the entire buffer. The routing decision initiated by packet 3 will be again $C(2,0)$.

This is exactly what we wanted to achieve in the standard algorithm by increasing d. We conclude that this mechanism is already present in the quick algorithm by the virtue of the fast removal of packets whose fate is known. Consequently, we cannot gain much by increasing d, but we lose some throughput due to the bandwidth wastage.

F	C(1,0)	C(2,0)	C(1,1)
0	0.214	0.783	0.003
0.5	0.314	0.682	0.004
1.0	0.399	0.598	0.003
1.5	0.467	0.531	0.002
2.0	0.518	0.481	0.001

Table 7.14: Quick algorithm with different F, load 20, $W = 1024$, $k = 2$

F	C(1,0)	C(2,0)	C(3,0)	C(4,0)	C(1,1)	C(2,1)	C(3,1)	C(1,2)	C(2,2)	C(1,3)
0	0.018	0.034	0.042	0.023	0.080	0.130	0.082	0.198	0.168	0.225
0.5	0.049	0.058	0.035	0.009	0.159	0.144	0.046	0.235	0.111	0.154
1.0	0.079	0.067	0.026	0.004	0.213	0.129	0.027	0.251	0.077	0.127
1.5	0.110	0.077	0.024	0.004	0.243	0.123	0.022	0.236	0.063	0.098
2.0	0.140	0.088	0.025	0.003	0.262	0.115	0.018	0.218	0.050	0.079

Table 7.15: Quick algorithm with different F, load 50, $W = 256$, $k = 4$

Tables 7.14 and 7.15 show the frequencies of the routing decision types for connectivities 2, and 4. For $k = 2$, increasing F (and thus d) has almost no effect on the number of worst routing scenarios $C(1,2)$. The routing decisions are not improved and the number of packets in the network decreases.

Throughput differences for $k = 4$ are smaller. This fact is also reflected in the number of the worst routing scenarios ($C(1,3)$) which drops much faster than for $k = 2$ (Table 7.15). This improvement counteracts the throughput deterioration caused by the smaller number of packets that may be inserted into the network.

7.4 The Complete Algorithm

This time we will try to improve the quality of a routing decision by postponing it until the last possible moment. With this approach, every packet is forced to go through the entire input buffer (as with the standard algorithm). Whenever a packet reaches the end of the input buffer and must be routed, the routing decision will take into account all packets currently visible to the switch, even if some of them have been already assigned to output ports by a previous routing decision.

This algorithm is complex. When the connectivity is high and input buffers are long, many packets may compete at the switch at the same time. Since solving the routing problem practically boils down to examining every possible assignment of packets to output ports, the algorithm may turn out to be infeasible for $k = 8$ or higher. This is why we confine our discussion to connectivities 2 and 4.

One might expect that the throughput achievable by this algorithm will never decrease with increasing W. The more packets are considered by the routing algorithm, the better its outcome should be. However, as shown in the tables in Section 7.2, a packet initiating a routing decision often finds all $k - 1$ ports busy. It has only one outgoing link to be routed over, which means that when the load is high (and gaps between packets are small), the routing algorithm is doomed to perform poorly regardless of the number of considered packets.

k=2			
W	algorithm	max.	saturated
1024	C	4.528	3.948
1024	S	7.212	5.543
1536	C	5.501	3.756
1536	S	6.471	5.349
2048	C	5.416	4.220
2048	S	5.687	4.991
k=4			
768	C	18.410	16.151
768	S	17.476	15.735
1024	C	18.911	15.413
1024	S	15.156	11.825
2048	C	16.249	13.341
2048	S	12.004	9.682

Table 7.16: Maximum and saturated throughput of the standard (S) and complete (C) algorithms for different W and k

As we can see in Table 7.16, the throughput in the network with $k = 2$ is even worse with the complete algorithm than with the standard one. To understand why, let us consider a scenario (in a connectivity-4 network) that is favorable for the complete algorithm (see Figure 7.8).

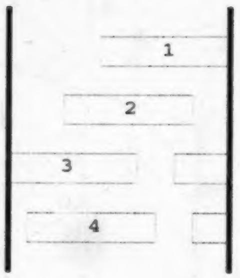


Figure 7.8: Complete algorithm, $k = 4$

In the standard algorithm, packet 1 would compete with packet 2 for two free ports. The routing decision would assign these two packets to the ports such that the cost C (combined deflection penalty) of this decision would be minimal. In the complete algorithm, packets 3 and 4 also take part in the routing decision. They may influence the assignment of packets 1 and 2 such that the cost of routing these two packets is higher than C but the cost of routing all four packets is reduced. This way, as more packets are considered, a better routing decision can be made.

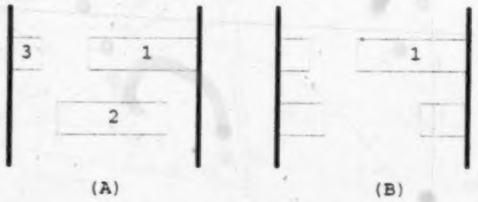


Figure 7.9: Complete algorithm, $k = 2$

Note, however, that this kind of improvement is only possible in a network with a connectivity higher than 2. If $k = 2$, only two scenarios can occur (Figure 7.9). In scenario (A), two packets compete for two free output ports. With the standard algorithm, the port assignment would be optimal from the viewpoint of packets 1 and 2. With the complete algorithm, including packet 3 in the routing decision might theoretically decrease the routing cost for all three packets by changing the assignment for 1 and 2. However, since packet 3 follows 1, 3 will only be routed after packet 1 leaves the buffer. This means that packet 1 can be routed over its best link without considering packet 3. Of course, the same situation occurs when 3 follows 2.

By the same token, in scenario (B), the number of packets considered in the routing decision does not matter. Packet 1 has only one port over which it can be routed.

Similar scenarios may occur in a network with connectivity 4. However, in such a network, the complete algorithm is at least given an opportunity to exhibit its feature. In some range of W, this opportunity outweighs the negative impact of the malicious scenarios.

The effect of increasing W is similar to what we have seen for the standard algorithm (Section 7.2), i.e., the throughput increases and then starts to decrease. It is noticed that the range of W in which the throughput increases is larger than in the standard algorithm.

W	C(1,0)	C(1,1)
1024	0.208	0.792
1536	0.225	0.775
2048	0.214	0.786

Table 7.17: Complete algorithm, load 14, k=2

W	C(1,0)	C(1,1)	C(1,2)	C(1,3)
768	0.029	0.150	0.405	0.416
1024	0.032	0.151	0.405	0.412
2048	0.024	0.128	0.393	0.455

Table 7.18: Complete algorithm, load 32, k=4

Tables 7.17 and 7.18 show the type frequency of routing decisions. Note that this time the assignment of packets to their outgoing ports is re-evaluated at every routing, and in fact only one packet is selected and forwarded to the free link. That is why we consider only k types of routing decisions: one packet competes when there are no busy links, one packet competes when there is one busy link, etc.

If we compare these tables with their counterparts in Section 7.2.1, we can see that for $k = 2$, the percentage of potentially bad decisions is higher in the complete algorithm than in the standard one. This is reversed for $k = 4$, in line with our earlier reasoning.

7.5 Comparison of Asynchronous and Synchronous Routing Algorithms

Table 7.19 compares all asynchronous routing algorithms presented in the preceeding sections. The waiting time W was chosen to maximize the throughput in each case.

As expected, the lowest throughput in the network with $k = 2$ is obtained when the complete algorithm is used. We showed in Section 7.4 that for connectivity 2, routing

k=2			
W	algorithm	max.	saturated
1024	Q	16.901	16.825
2048	C	5.416	4.220
1024	S,F=2.0	8.157	8.049
1024	S	7.212	5.543
k=4			
1024	Q	42.797	42.193
768	C	18.911	15.413
256	S,F=2.0	21.286	20.721
256	S	18.605	15.456
k=8			
512	Q	132.436	132.124
128	S,F=1.0	77.935	77.933
128	S	72.957	72.757

Table 7.19: Maximum and saturated throughput for different W and k—comparison of asynchronous schemes (S-standard, Q-Quick, C-Complete)

decisions can only be worsened by considering all packets competing at the switch.

A higher throughput is achieved by the standard algorithm, particularly if the delay line is longer than L. We showed in Section 7.2.1 that increasing the gaps between the packets improves the routing decisions made by this algorithm.

Table 7.20 compares the maximum throughput in asynchronous and synchronous torus networks. The amount of buffer space in the synchronous networks was equal to the minimum required for slot alignment, i.e., one slot per input port.

We may see that the throughput in the synchronous networks exceeds the throughput in their asynchronous counterparts, even if the best (quick) algorithm is used. When the load is sufficiently high, some packets will always be "stuck" behind the packets whose fate has been already determined. Their chances for being routed via their prefered links are smaller because there are fewer free output ports to choose from.

Visible as it is, the difference between the quick algorithm and synchronous routing is much smaller than between the standard algorithm and the quick one. This difference amounts to about 7% for $k = 2$, 19% for $k = 4$, and 25% for $k = 8$.

7.6 Summary

Our results indicate that among the asynchronous routing algorithms, the best are those that assure that a packet competing at a switch will perceive a large number of free ports.

k=2			
W	algorithm	max.	saturated
	synch.	18.306	
1024	Q	16.901	16.825
2048	C	5.416	4.220
1024	S,F=2.0	8.157	8.049
k=4			
	synch.	52.386	
1024	Q	42.797	42.193
768	C	18.911	15.413
256	S,F=2.0	21.286	20.721
k=8			
	synch.	165.838	
512	Q	132.436	132.124
128	S,F=1.0	77.935	77.933

Table 7.20: Maximum and saturated throughput for different W and k—comparison with synchronous routing

That is why, the standard algorithm with long delay lines and the quick algorithm give the best results.

We have also shown that some solutions may produce counterintuitive results, e.g., increasing the amount of buffer space may deteriorate the network's performance. For this reason, the selection of the buffer size in an asynchronous network should be made with caution. Generally, the higher the connectivity, the smaller the recommended length of the buffer.

Although our experiments have been carried out for Poisson traffic, one should expect that the performance of the routing algorithms for other traffic patterns will be similar. As we have noticed in Chapter 2, deflection networks are not very sensitive to changing traffic patterns.

Among the routing algorithms discussed in this chapter, the standard algorithm is the simplest, and the quick algorithm (which is only slightly more complicated) offers the highest throughput. This throughput approaches that achievable with synchronous routing; thus, asynchronous deflection appears to be a feasible alternative, especially in high-speed applications, where synchronous deflection may be difficult to implement.

Chapter 8

Deflection Networks vs Store-and-Forward Networks

So far, we have shown that deflection networks posses features that make them suitable in high speed environments. In this chapter, we compare the performance and buffer space requirements in deflection and store-and-forward networks. We show that if no resources are reserved in advance, a deflection network performs much better than a store-and-forward network with a similar set of resources.

First, we present the models of both networks. Then, we discuss simulation results of two video applications. Finally, we conclude this chapter.

8.1 Network Model

8.1.1 Deflection Network

Simulation model of deflection networks used in this chapter is identical to the model presented in Section 4.1. That is, we look at the performance of a video session between a selected pair of switches. Also, the receiver may sometimes be resynchronized with the transmitter, as it was described in Section 4.2.1.

8.1.2 Store-and-forward Network

The model of a store-and-forward network operates as follows. Every packet, upon entering a switch, is stored in an intermediate buffer.[1] There is one such buffer in every switch. After all incoming packets are admitted to the switch, the switch may insert its own packets at the end of the intermediate buffer if it has has anything to transmit and if there is free space in the buffer. Note that the number of packets that may be transmitted in one routing cycle is greater than one. It is limited only by the available free space in the buffer and the number of packets that the switch wants to transmit. In other words, the sender tries to insert as many packets as possible, filling completely the buffer. This allows it to obtain a better access time (and better overall delay).

---

[1] As in the model of deflection networks, it is first stored in a short, one-slot buffer for alignment.

Then, the packet from the buffer's head is routed to its best link. This link is determined on the basis of the route stored in the packet's header. Then, the whole buffer is scanned in search for packets whose next switch on their paths may be reached through one of the remaining free links. If such packets are found, they are routed to their preferred links. Note that in this way, up to k (where k is the network connectivity) incoming packets may be relayed to outgoing links in one routing cycle. It greatly improves the performance because packets belonging to different sessions do not block each other if they are to be relayed over different outgoing links. At the same time, packets belonging to the same session are not misordered. Packets that remained in the buffer after this operation are shifted to the head of the buffer. This completes the routing cycle.

In the algorithm presented above, incoming packets may be dropped. In every routing cycle, up to k packets are inserted to the buffer. However, if for example all packets in the buffer belong to the same session and they are supposed to be routed over the same outgoing link, only one of them may be removed from the buffer. The number of incoming packets may thus exceed the number of relayed packets which causes a buffer overflow and packet loss. In deflection networks, this could not happen because every packet had to be relayed over some outgoing link even if this link was not preferred.

If incoming packets are considered one by one always starting from port 0 up to port $k-1$, it is possible that packets arriving from higher numbered links will be constantly dropped. This will happen for example if only one packet is removed from the buffer in one routing cycle, and more than one packet arrives to the switch. Clearly, packets from link 0 will be entering the buffer while packets arriving from other links will be dropped. For this reason, the order with which packets are fetched from incoming ports is randomized. It gives (on average) the same chances to every session passing through the switch.

Packets always arrive to the destination in the order of their transmission. Packets belonging to the same session always follow the same route determined when the session is initiated. This route is determined such that the distance from the source to the destination is the shortest. That is why a packet's path between an active pair of switches is always the same.

Synchronization, as introduced in Section 4.2.1, is very helpful in deflection networks when no packets are dropped at intermediate switches. A small number of packets received from a given frame suggests that packets were delayed and the receiver lost synchronization with the sender. Some frames are played before all packets belonging to this frame appear of the receiver.

In a store-and-forward network, packets may fail to arrive to the receiver because they were dropped at some intermediate switch. Resetting the session, which always involves losing some number of already received packets, may only make the situation worse. Of course, it is possible that the awaited packets are not dropped but simply delayed like in deflection networks (which may happen when the sender is blocked by other sessions). In this case, reset and resynchronization would be beneficial. However, the receiver does not know what causes the delay and for this reason, as our simulations indicate, in most cases it is safer not to allow the receiver to reset any sessions.

We have observed, however, that if the size of intermediate buffers is large enough to assure a small packet loss at intermediate switches, the packet loss obtained from the simulator with synchronization is lower. These results, shown in the tables in Section 8.2, are denoted with *. They were obtained from the simulator with synchronization. We are not concerned here *how* the receiver found out that synchronization is beneficial in the given network (we have mentioned that it would be rather difficult). We simply wish to present the best results obtained from the simulations of our store-and-forward network.

In the next section, we show a comparison between the deflection and store-and-forward network in the environments characteristic to videophone and transmission of video movies.

8.2 Results

8.2.1 Videophone Traffic

This environment was introduced in Section 4.2.1.

In the deflection network, we were changing size B of the playout buffer and investigating the average packet loss and end-to-end delay. In the store-and-forward network, we set the playout buffer size to $B = 2$ frames. Then, we were changing the buffer size at intermediate switches and observing the performance measures. We also investigated the overall buffer space allocated in the network, excluding 1-slot buffers for alignment and transmitter queues (whose size in both networks were identical).

buffer space (frames)	loss	delay (s)
4.00	0.0071	0.1598
6.00	0.0047	0.1969
8.00	0.0014	0.1618
10.00	0.0007	0.1953
12.00	0.0011	0.2439
14.00	0.0003	0.2359

Table 8.1: Deflection network in videophone application (k=2)

intermediate buffer space (slots)	buffer space (frames)	loss	delay (s)
64	10.51	0.3385	0.0771
128	17.02	0.2251	0.0798
256	30.04	0.1306	0.1035
512	56.08	0.1248	0.1100
1024 *	108.17	0.1094	0.1343
2048 *	212.34	0.1092	0.1930

Table 8.2: Store-and-forward network in videophone application (k=2)

Tables 8.1 and 8.2 show the total buffer space expressed in frames[2], packet loss and end-to-end delay in the deflection and store-and-forward networks, respectively.

The first column in Table 8.1 shows the sum of buffer spaces at switches $S0$ and $S1$. Thus, e.g., 4.0 means that both switches are equipped with playout buffers of size 2.0 frames. Note that the playout buffer in a deflection network is the only buffer space allocated in the network, needed by the selected pair of switches.

The first column in Table 8.2 shows the size of the intermediate buffer in slots at every switch. The second column shows the overall buffer size in the entire network including the playout buffers in $S0$ and $S1$ expressed in frames[3]; * denotes results obtained from the simulator with synchronization. In those cases, the packet loss at intermediate buffers was smaller than the packet loss in playout buffers. It means that the packet loss was caused mainly by the fact the receiver was not properly synchronized with the sender. Synchronization significantly improved the performance.

We may observe a huge difference in packet loss between the deflection and store-and-forward network. We have noticed that packets in the latter one are lost mainly at intermediate buffers which means that playout buffers are sufficiently large. Under these conditions, packet loss in store-and-forward network is thus caused by the insufficient amount of buffer space at intermediate switches.

---

[2] With the parameters listed in Section 4.2.1, one frame in the 2-connected network contains 983 packets (or slots), and one frame in the 4-connected network contains 492 packets.

[3] For example, buffer space of $10.51 = (64 * 100)/983 + 4$.

Note that at the same time, total buffer space requirements are much smaller in the deflection network. On the other hand, end-to-end delay in the 2-connected deflection network is about twice as large as the delay in the store-and-forward network.

This difference in delays is caused by much lower access delays at the sources in the store-and-forward network. In such a network, packets are blocked in sender S only by the packets whose sessions were initiated such that their paths are passing through S. Usually, there are periods when S is blocked (a relevant source of a background session is active with frame transmission), and long periods when S may transmit freely (a relevant source already transmitted all packets belonging to a frame). Of course, if there are more than one background session passing through S, these periods may be less regular. The access time is also reduced because a sender may insert to the buffer many (up to the intermediate buffer size) packets in one routing cycle.

In the deflection network, a sender may transmit only up to k packets per routing cycle. Moreover, access to the network is less regular. Various packets from many sources may pass through S, deflected from their shortest paths. Even if a background source stops transmitting for a moment, there is a probability that some other packets will take the place of the suddenly "missing" packets. Moreover, packets originating from S may return to S further throttling its traffic and increasing access delay—something that cannot happen in a store-and-forward network. It all has a strong impact on access time to the network.

However, we can imagine a situation in which many sessions pass through some source in a store-and-forward network. In such a case, this source will be usually blocked. In a deflection network, even under heavy load, there is always a possibility that the source will perceive a free slot allowing it to transmit its packet.

We have observed that packets in a store-and-forward network are rarely dropped in the playout buffer at the receiver even if its size is small (2 frames). Since many packets are dropped at intermediate buffers, more frames on average fit in the playout buffer. It is in fact equivalent to increasing a playout buffer size. Those packets that make it to the receiver tend to arrive there in a rather regular fashion (usually one by one in every consecutive cycle).

In a deflection network, irregularity of transmissions and arrivals (caused by the fact that packets follow different paths) makes the receivers drop packets relatively often. Playout buffers are sometimes too short.

buffer space (frames)	loss	delay (s)
4.00	0.0046	0.0827
6.00	0.0016	0.1004
8.00	0.0000	0.1337

Table 8.3: Deflection network in videophone application (k=4)

intermediate buffer space (slots)	buffer space (frames)	loss	delay (s)
256 *	56.03	0.5162	0.1093
512 *	108.06	0.5150	0.1544
1024 *	212.13	0.4814	0.2496

Table 8.4: Store-and-forward network in videophone application (k=4)

Of course, as we have already mentioned, it may happen that packets originating from a large number of sources in store-and-forward networks will have paths passing through the monitored source blocking it completely. In such a case, we would observe more packets dropped in the playout buffer.

This seems to be confirmed by our results shown in tables 8.3 and 8.4. In the 2-connected network, there were 34 pairs of active background switches and we observed a slight packet loss in the playout buffer only when the total buffer space was the largest. Clearly, it was caused by the fact that fewer packets were dropped at intermediate switches.

In the 4-connected network, there are 42 pairs of active background switches causing some packet loss in the playout buffer regardless of the amount of total buffer space. More (than in the 2-connected network) paths are likely to include the observed source and, on the average, more packets may be blocked at this source. More packets arrive to the destination irregularly what causes a higher packet loss in the playout buffer. The fact that other switches on the paths from the selected source are also more loaded further increases the packet loss. To improve the results, we applied the algorithm with synchronization.

The propagation delay (and thereby end-to-end delay) is also higher in the network in which more switches are active. In the 2-connected network, a packet rarely has to pass through a large buffer in every intermediate switch. In the example 4-connected network, due to the greater number of active sessions, packets usually have to spend more time in intermediate switches. This delay obviously increases with the increasing buffer space.

We have also performed simulations with a larger playout buffer. The packet loss was similarly disastrous. This indicates that the large number of active sessions in the network causes frequent buffer overflows and a high packet loss. There seems to be a strong correlation between the number of active sessions and the performance of a store-and-forward network.

8.2.2 Transmission of Video Movies

On the basis of our observation from Chapters 3 and 4, we consider two multicast algorithms: R_Class or R_Ext. We have observed that in the transmission of video movies, the application of R_Ext with the minimum replication distance $R_d = 1$ gives much better results than application of R_Class (or R_Ext with large R_d). Every replication in R_Class may produce several M-packets that will further divide in subsequent switches. We have observed that when the number of multicast destinations is large, a single M-packet sent by the source may produce many multicast fragments that will circulate in the network. These fragments may inhibit not only other switches but the source of the M-packet itself.

In a store-and-forward network, M-packets belonging to a given session are always replicated at the same switches and they always follow the same paths to their destinations. They can neither return to the source blocking new transmissions nor deflect other packets from their routes. When an M-packet is replicated and some of its fragments overflow the intermediate buffer, these fragments are dropped. The order in which they are dropped is random. This way, no destination is being privileged.

At first sight, dropping fragments addressed to a smaller number of destinations seems to be a good idea. This way, packets addressed to a large number of destinations would be more likely to "survive" which would probably increase the overall throughput of the multicast traffic. However, it could also decrease the throughput (and increase the jitter) perceived by destinations located far from the source. Fragments of M-packets addressed to such destinations usually carry very few destinations in their headers and they would be most affected by overflows of intermediate buffers. For this reason, as mentioned above, a fragment to be dropped is picked at random.

A multicast source no longer inserts to the network as many packets as possible, like it is done in the videophone traffic. Instead, only one packet per routing cycle may be transmitted. This is because intermediate buffers should not be full. If a buffer is full at some switch and an M-packet is to be replicated at this switch, it is obvious that its fragments will have to be dropped. For this reason, it is better to somewhat limit the transmission rate of a multicast source.

Thus, in the simulations of a deflection network we use the R_Ext scheme with the minimum replication distance $R_d = 1$, and in the simulations of a store-and-forward network we use R_Class.

buffer space (frames)	loss	delay (s)
2.00	0.0001	0.0649
3.00	0.0000	0.0990
4.00	0.0000	0.1290
5.00	0.0000	0.1659
6.00	0.0000	0.1992

Table 8.5: Deflection network in transmission of video movies (k=2)

intermediate buffer space (slots)	buffer space (frames)	loss	delay (s)
128	15.02	0.0271	0.0680
256	28.04	0.0167	0.0683
512	54.08	0.0068	0.0683
1024 *	106.17	0.0035	0.0695
2048 *	210.34	0.0027	0.0820

Table 8.6: Store-and-forward network in transmission of video movies (k=2)

buffer space (frames)	loss	delay (s)
2.00	0.0000	0.0640
3.00	0.0000	0.0974
4.00	0.0000	0.1307
5.00	0.0000	0.1641
6.00	0.0000	0.1980

Table 8.7: Deflection network in transmission of video movies (k=4)

We have compared (Tables 8.5–8.8) the performance of a deflection network using the best multicast algorithm in this application (*R_Ext* with replication distance $R_d = 1$) with the performance of a store-and-forward network (*R_Class* algorithm). Note that the playout buffer space per destination is considered, not the sum of sizes of all playout buffers in all $D = 30$ destinations. As before, * denotes results obtained from the simulator with synchronization.

The results are in fact similar to what we saw in the context of the videophone traffic. The performance of the store-and-forward network is much worse than the performance of the deflection network despite the fact that buffer space used by the former is much greater.

It seems that without any advanced reservation techniques, both the point-to-point and

intermediate buffer space (slots)	buffer space (frames)	loss	delay (s)
128	15.02	0.0625	0.0678
256	28.04	0.0206	0.0689
512 *	54.08	0.0410	0.0793
1024 *	106.17	0.0314	0.0869
2048 *	210.34	0.0285	0.0963

Table 8.8: Store-and-forward network in transmission of video movies (k=4)

multicast applications perform much worse in the store-and-forward network than in the deflection network.

8.3 Summary

It seems that increasing the intermediate buffer space in a store-and-forward network, although it somewhat decreases packet loss, is not a good solution. As it was pointed out, every intermediate buffer may finally overflow. It depends on the network load and existing traffic patterns in the network. What makes things worse is the fact that it is harder (or even impossible) to control their sizes.

Of course, the same can be said about deflection networks. However, it is easy to change the size of a playout buffer if it is needed (see Chapter 6). Moreover, although the jitter is theoretically unbounded, large deviations from the average interarrival delay are very rare and therefore it is possible to dynamically set a playout buffer that will assure an acceptable packet loss.

Desired performance in a store-and-forward network may be achieved only when resources are reserved in advance. Without any such mechanisms, performance of a "pure" store-and-forward network is inferior to the performance of a "pure" deflection network.

Chapter 9

Conclusions and Suggestions for Future Research

9.1 Conclusions

In this dissertation, we have studied the suitability of the connection-less paradigm, exemplified by deflection routing, in high-speed networks.

We have presented experimental results showing the expected jitter, and consequently buffer space requirements, in deflection networks used to carry traffic with timing constraints. We have concluded that with reassembly buffers of a reasonable size, the network can cater to practically any sensible isochronous session, offering a very low packet loss rate. Moreover, these buffers are allocated on a per session basis exclusively at the destination.

We have pointed out that multicasting is also important in high-speed networking environments. We have presented a number of simple multicast schemes for deflection networks. Some of those schemes use limited replication, which does not depend on extra buffer space at a switch. Notably, among the simpler schemes, the one involving the least amount of processing turns out to be the best. The proposed schemes seem to be especially well suited for relatively small multicast groups because the length of an M-packet's header increases with the increasing group size. However, this increase may be reduced by partitioning the multicast recipients.

Since the jitter is not large and the reassembly buffer of reasonable size can easily smooth it out, we could expect that deflection networks may perform well in isochronous applications. Also, thanks to the proposed multicast schemes, we could test the performance of the networks in applications like videoconference that require this service from the network. We have observed the performance of three multimedia applications in deflection networks. We have proposed an efficient and simple synchronization scheme that significantly reduces the packet loss, particularly when the playout buffer is too small or when the conditions in the network temporarily disrupt the regularity with which packets arrive at the receiver. We have also examined the suitability of different multicasting schemes in videoconference and transmission of video movies. Our results suggest that despite the fact that packets may arrive at the receivers misordered, the performance of deflection networks in multimedia applications turns out to be very good.

We have also investigated the issue of quality of service in deflection networks. We have presented a protocol implementing simple, statistical measures and investigated its performance under several types of traffic conditions ranging from Poisson to bursty. Simulation results indicate that the network performance for synchronous and isochronous traffic remains stable, even under relatively heavy datagram loads. The performance for datagram traffic decreases, but the total decrease in the maximum throughput is small. The increase of jitter is also very small, which suggests usefulness of this algorithm in jitter-sensitive applications. The protocol works efficiently even when the network topology is irregular (triangle) and a large fraction of all stations participate in synchronous sessions. Our solution does not require any pre-reservation of resources, and no bandwidth is wasted on token exchange.

Our next approach to this problem provides a long-term quality of service in the deflection network. We have obtained this by limiting the number of active sources coupled with the receivers' ability to adjust the sizes of their playout buffers. Similar to our previous solution, no resources are reserved in advance and no bandwidth is wasted on tokens. This also suggests that this protocol is easily scalable. In fact, with the increasing size of the network and/or network's capacity, mainly the setup phase, in which sources initiating a session must collect confirmations from each other, has to be extended.

Next we have studied several asynchronous routing algorithms. Asynchronous deflection poses an interesting (and more realistic) alternative to slotted synchronous deflection networks. We have noticed that the best algorithms are those that assure that a packet competing at a switch will perceive a large number of free ports. We have also shown that some solutions may produce counterintuitive results, e.g., increasing the amount of the buffer space may deteriorate the network's performance. For this reason, the selection of the buffer size in the asynchronous network should be made with caution. With buffers of the appropriate size, the throughput approaches that achievable by synchronous routing; thus, the asynchronous deflection appears to be a feasible alternative, especially in high-speed applications, where the synchronous deflection may be difficult to implement.

Finally, we have compared the performance of a deflection network with the performance of a store-and-forward network. It seems that increasing intermediate buffer space in a store-and-forward network, although it somewhat decreases packet loss, is not a good solution. Intermediate buffers may finally overflow and what makes things worse is the fact that it is harder (or even impossible) to control their sizes. The desired performance of a store-and-forward network may be achieved only when the network resources are reserved in advance. Without such mechanisms, the performance of a "pure" store-and-forward network is inferior to the performance of a "pure" deflection network.

From our studies of deflection networks, we conclude that the application of a connectionless paradigm in high-speed networks is a feasible alternative to a connection-oriented paradigm.

9.2 Future Research

9.2.1 Distributed Computing

We have not investigated the performance of deflection networks under scenarios characteristic to distributed computing. These scenarios will be certainly common in high speed networks. One of their features, present also in other high-speed applications, is the high burstiness of the traffic they introduce. We have shown that deflection networks perform very well even under "unfriendly" traffic scenarios, including bursty ones. We may thus expect that their performance will be satisfactory also in distributed computing. However, it would be interesting to see their performance in some specific applications.

9.2.2 Mobile Computing

Mobile computing gains more and more significance. Computer users want to access their file systems or read e-mail regardless of their geographical location. Mobile computing poses new problems such as the disparity between the low bandwidth of a wireless medium (usually associated with mobile computing) and the high bandwidth of a "stationary" network. Routing in such networks is also more difficult due to their dynamic nature.

Mobile networks are highly dynamic—a user may quickly change the cells changing the foreign agents it communicates with. With a connection-oriented approach every transition from one cell to another could involve tearing down the old connection and establishing the new one which is obviously expensive.

Very high routing flexibility of deflection networks fits in this challenging environment. There is no notion of circuit or stream present in the network and packets are routed individually. This suggests the suitability of deflection networks in mobile computing and it would be interesting to investigate it.

9.2.3 Comparison with Other Networks

In Chapter 8, we have compared the performance of a deflection network with a performance of a generic store-and-forward network. It would be interesting to make similar comparisons with some specific networks under the specific traffic scenarios that we can expect to find in high-speed networks. Example comparison with DQDB has been made by authors of [MK93] who have shown that the performance of a deflection network is better than the performance of DQDB.

9.2.4 Further Studies of the Quality of Service

We have proposed a scheme improving the performance of a stream traffic regardless of the intensity of the datagram traffic (Chapter 5), as well as the protocol allowing to achieve a long term quality of service (Chapter 6). In the context of this protocol, we have considered only isochronous (video) sessions. It would be interesting to merge these two schemes and investigate the performance of a network under mixed (datagram and stream) traffic scenarios. The first scheme could throttle the excessive datagram traffic while the latter one would control the admission of new stream sources to the network and provide a quality of service.

Bibliography

[Bar64] P. Baran. On distributed communication networks. *IEEE Transactions on Communications*, 12:1–9, March 1964.

[BC90] F. Borgonovo and E. Cadorin. Locally-optimal deflection routing in the bidirectional Manhattan network. In *Proceedings of IEEE INFOCOM'90*, pages 458–464, 1990.

[BDG95] C. Baransel, W. Dobosiewicz, and P. Gburzynski. Routing in multi-hop switching networks: Gbps challenge. *IEEE Network Magazine*, (3):38–61, 1995.

[BF92] F. Borgonovo and L. Fratta. Deflection networks: architectures for metropolitan and wide area networks. *Computer Networks and ISDN Systems*, (24):171–183, 1992.

[BO98] M. Baldi and Y. Ofek. End-to-end delay of videoconferencing over packet switched networks. In *Proceedings of IEEE INFOCOM'98*, pages 1084–1092, 1998.

[BSTW95] J. Beran, R. Sherman, M.S. Taqqu, and W. Willinger. Long-range dependence in variable-bit-rate video traffic. *IEEE Transactions on Communications*, 3:1566–1579, 1995.

[Cho91] A.K. Choudhury. *Deflection Routing in High-Speed Networks*. PhD thesis, University of Southern California, August 1991.

[CL91] A.K. Choudhury and V.O.K. Li. Effect of contention resolution rules on the performance of deflection routing. In *Proceedings of the IEEE Conference on Global Communications (GLOBECOM)*, pages 1706–1711, Phoenix, Arizona, December 1991.

[CM91] A.K. Choudhury and N.F. Maxemchuk. Effect of a finite reassembly buffer on the performance of deflection routing. *Conference Record of the International Conference on Communications (ICC)*, 3:1637–1646, 1991.

[CS92] R. Cohen and A. Segall. Multiple logical token rings in a single high-speed ring. *Technion Technical Report #738*, 1992.

[DBH96] S. Deng, A.L. Bugos, and P.M. Hill. Design and evaluation of an Ethernet-based residential network. *IEEE Journal on Selected Areas in Communications*, 14(6):1138–1150, aug 1996.

[DC90] S.E. Deering and D.R. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990.

[DG93] W. Dobosiewicz and P. Gburzyński. On token protocols for high-speed multiple-ring networks. In *Proceedings of the International Conference on Network Protocols*, pages 300–307, San Francisco, USA, October 1993.

[DG94] W. Dobosiewicz and P. Gburzyński. An alternative to FDDI: DPMA and the pretzel ring. *IEEE Transactions on Communications*, 42:1076–1083, 1994.

[DG96] W. Dobosiewicz and P. Gburzynski. A bounded-hop-count deflection scheme for Manhattan-street networks. In *Proceedings of IEEE INFOCOM'96*, San Francisco, March 1996.

[DQD91] Distributed Queue Dual Bus Subnetwork of a Metropolitan Area Network. IEEE Std. 802.6–1990, July 1991.

[FF62] A.J. Ford and D.R. Fulkerson. *Flows in networks*. Princeton University Press, 1962.

[FMO+91] E.C. Foudriat, K. Maly, C.M. Overstreet, S. Khanna, and F. Paterra. A Carrier Sense Multiple Access protocol for high data rate ring networks. *ACM Computer Communication Review*, 21(2):59–70, April 1991.

[GG86] A.G. Greenberg and J. Goodman. Sharp approximate models of adaptive routing in mesh networks. In O.J. Boxma, J.W. Cohen, and H.C. Tijms, editors, *Teletraffic Analysis and Computer Performance Evaluation*, pages 255–270. Elsevier Science Publishers B.V. (North-Holland), 1986.

[GH92] A.G. Greenberg and B. Hajek. Deflection routing in hypercube networks. *IEEE Transactions on Communications*, 40(6):1070–1081, June 1992.

[GHM+91] J.N. Giacopelli, J.J. Hickey, W.S. Marcus, W.D. Sincoskie, and M. Littlewood. Sunshine: a high-performance self-routing broadband packet switch architecture. *IEEE Journal on Selected Areas in Communications*, 9(8):1289–1298, October 1991.

[GKV+90] M.S. Goodman, H. Kobrinski, R.M. Vecchi, R.M. Bulley, and J.L. Gimlett. The LABMDANET multiwavelength network: Architecture, applications, and demonstrations. *IEEE Journal on Selected Areas in Communications*, 8(6):995–1004, August 1990.

[GLH90] D.J. Greaves, D. Lioupis, and A. Hopper. The Cambridge Backbone Ring. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, pages 8–14, San Francisco, California, June 1990.

[GM93] P. Gburzynski and J. Maitan. Deflection routing in regular MNA topologies. *Journal of High Speed Networks*, 2(2):99–131, 1993.

[GW94] M.W. Garret and W. Willinger. Analysis, modeling and generation of self-similar VBR video traffic. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 269–280, London, UK, September 1994.

[HRL95] H-Y Huang, T. Robertazzi, and A.A. Lazar. A comparison of information based deflection strategies. *Computer Networks and ISDN Systems*, 27:1399–1407, 1995.

[JR86] R. Jain and S. Routhier. Packet trains: measurement and a new model for computer network traffic. *IEEE Journal on Selected Areas in Communications*, 4(6):1162–1167, May 1986.

[Kam90] A. Kamal. On the use of multiple tokens on ring networks. In *Proceedings of IEEE INFOCOM'90*, pages 15–22, San Francisco, CA, June 1990.

[KH90] A. Krishnan and B. Hajek. Performance of shuffle-like switching networks with deflection. In *Proceedings of IEEE INFOCOM'90*, pages 473–480, San Francisco, CA, June 1990.

[KS92] H.T. Khalil and Y.S. Sun. On the performance of protocols for collecting responses over a multiple-access channel. In *Proceedings of the IEEE Conference on Global Communications (IEEE GLOBECOM)*, pages 597–603, Orlando, Florida, December 1992.

[Kun92] H.T. Kung. Gigabit local area networks: A systems perspective. *IEEE Communications Magazine*, 30:79–89, April 1992.

[LB92] J.Y Le Boudec. The Asynchronous Transfer Mode: a tutorial. *Computer Networks and ISDN Systems*, 24:279–309, 1992.

[Lie95] S.C. Liew. A general packet replication scheme for multicasting in interconnection networks. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, pages 394–401, Boston, Massachusetts, April 1995.

[LTWW94] W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson. On the self-similar nature of Ethernet traffic. *IEEE Transactions on Networking*, 2(1):1–15, February 1994.

[Max85] N.F. Maxemchuk. The Manhattan Street Network. In *Proceedings of GLOBECOM'85*, pages 255–261, 1985.

[Max87] N.F. Maxemchuk. Routing in the Manhattan Street Network. *IEEE Transactions on Communications*, 35(5):503–512, May 1987.

[Max88] N.F. Maxemchuk. Distributed clocks in slotted networks. In *Proceedings of IEEE INFOCOM'88*, pages 119–125, 1988.

[Max89] N.F. Maxemchuk. Comparison of deflection and store-and-forward techniques in Manhattan-street network and shuffle-exchange networks. In *Proceedings of IEEE INFOCOM'89*, pages 800–809, 1989.

[Max91] N.F. Maxemchuk. Problems arising from deflection routing. In Pugolle, editor, *High Capacity Local and Metropolitan Networks*, pages 209–233. Springer Verlag, 1991.

[MB76] R.M. Metcalfe and D.R. Boggs. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM*, 19(7):395–404, July 1976.

[MK93] N. Maxemchuk and R. Krishnan. A comparison of linear and mesh topologies-DQDB and the Manhattan Street Network. *IEEE Journal on Selected Areas in Communications*, 11(8):1278–1301, October 1993.

[MLNP93] M. A. Marsan, E. Leonardi, F. Neri, and G. Pistritto. Service guarantees in deflection networks. In *6th IEEE Workshop on Local and Metropolitan Area Networks*, San Diego, California, October 1993.

[Mog91] J.C. Mogul. Network locality at the scale of processes. In *ACM SIGCOMM'91*, pages 273–284, Zurich, September 1991.

[MRR80] J.M. McQuilan, I. Richer, and E.C. Rosen. The new routing algorithm for the ARPANET. *IEEE Transactions on Communications*, 28:711–719, May 1980.

[MWW90a] J. Maitan, L. Walichewicz, and B Wealand. A new low cost communication scheme for military applications. In *Proceedings of Milcom'90*, Monterey, CA, 1990.

[MWW90b] J. Maitan, L. Walichiewicz, and B. Wealand. Integrated communication and information fabric for space applications. In *AIAA/NASA Second International Symposium on Space Information Systems*, pages 1175–1184, September 1990.

[OG] W. Olesinski and P. Gburzynski. Service guarantees in deflection networks. *Telecommunications Systems*. (Submitted for publication).

[OG98a] W. Olesinski and P. Gburzynski. Asynchronous deflection with transient buffers. In *Proceedings of the Seventh International Conference on Computer Communications and Networks (IC3N'98)*, Lafayette, Louisiana, October 1998.

[OG98b] W. Olesinski and P. Gburzynski. Multicast in deflection networks. In *Proceedings of Sixth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'98)*, pages 50–55, Montreal, Canada, July 1998.

[OG98c] W. Olesinski and P. Gburzynski. Quality of service in deflection networks. In *Proceedings of 9th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN'98)*, pages 11–16, Banff, Canada, May 1998.

[OG98d] W. Olesinski and P. Gburzynski. Real-time traffic in deflection networks. In *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation (CNDS'98)*, pages 23–28, San Diego, California, January 1998.

[Par94] C. Partridge. *Gigabit Networking*. Addison-Wesley, 1994.

[RK63] R.R. Riesz and E.T. Klemmer. Subject evaluation of delay and echo suppressors in telephone communications. *The Bell System Technical Journal*, 42:2919–2941, November 1963.

[RN96] B.K. Ryu and M. Nandikesan. Real-time generation of fractal ATM traffic: model, algorithm, and implementation. Technical report 440-96-06, Department of Electrical Engineering and Center for Telecommunication Research, Columbia University, New York, NY, March 1996.

[Ros89] F.E. Ross. An overview of FDDI: The Fiber Distributed Data Interface. *IEEE Journal on Selected Areas in Communications*, 7(7):1043–1051, September 1989.

[Tan96] A.S. Tanenbaum. *Computer Networks*. Prentice-Hall, Englewood Cliffs, New Jersey, 1996. Third edition.

[TR93] D. Tolmie and J. Renwick. HIPPI: Simplicity yields success. *IEEE Network Magazine*, 7(1):28–33, January 1993.

[Tur86] J.S. Turner. New directions in communications. *IEEE Communications Magazine*, 24(10):6–15, October 1986.

[WLS97] M.H. Willebeek-LeMair and Z-Y Shae. Videoconferencing over packet-based networks. *IEEE Journal on Selected Areas in Communications*, 15(6):1101–1114, August 1997.

[YHA87] Y-S. Yeh, M.G. Hluchyj, and A.S. Acampora. The Knockout Switch: a simple, modular architecture for high-performance packet switching. *IEEE Journal on Selected Areas in Communications*, 5(8):1274–1283, oct 1987.

Appendix A

Control messages

Control messages used in the protocol presented in Chapter 6.

SD (slow-down) request.

Sent by source S to the source in the setup phase whose packets most frequently inhibit S's transmission.

CC (connection confirm) message.

Sent by the receiver to the source that initiates the session. It informs the source in a setup phase that a required quality of service may be sustained.

CT (connection terminate) message.

Sent by source S to the receiver and to peer $\hat{S}$ during the setup phase. It indicates that a packet queue at S increases or that S received SD request from the active source. It leads to blocking a session.

CA (connection accept) message.

Sent by source S to peer $\hat{S}$. It indicates that S is ready to begin the active phase of a session.

CF (connection finished) message.

Sent by the source to the receiver. It completes the session.

Appendix B

The Simulator

We have used two types of simulator to obtain the results presented in this thesis: synchronous (slotted) and asynchronous. Results given in Chapters 2 - 6 and 8 were produced by the slotted simulator while the results in Chapter 7 were produced by the asynchronous simulator.

Below, we describe the general structure of both simulators.

B.1 Slotted Simulator

The kernel of this simulator was written in C++ by Prof. Pawel Gburzynski. Its correctness was verified in [GM93]. In that paper, the authors used this simulator to obtain the throughput of a deflection network and compared it to the throughput obtained from theoretical analysis.

We have extended the slotted simulator to be able to simulate various protocols presented in this thesis and investigate diverse performance measures relevant in the context of a particular protocol. We have also written and used a number of traffic generators like Poisson, bursty, self-similar to be able to observe the behaviour of the deflection network under various traffic conditions.

Let the switch connectivity (i.e., the number of incoming and outgoing links at a switch) be k. Every switch in the simulated network is represented by a C++ object consisting of the following main fields:

- `Id` — the numerical identifier of the switch;
- `Neighbour` — array with k Ids of the immediate neighbours;
- `Incoming` — array representing k incoming ports of the switch. A packet (slot) arrives from the neighbour `Neighbour[i]` to the port `Incoming[i]`, where $0 \leq i < k$.

- `Outgoing` — array representing k outgoing ports of the switch. A packet that leaves the switch from the outgoing port `Outgoing[i]` will be received by `Neighbour[i]`.

- `Link` — 2-D array representing k links outgoing from the switch. A packet leaves the switch through port `Outgoing[i]` and is inserted to link `Link[i]`. If the length of the link is l, the packet arrives to the neighbour `Neighbour[i]` l slots (or units of time) later. It is then inserted to the appropriate port of the neighbour's array `Incoming`.

- `Buffer` — array of slots representing the buffer that gives the switch enough time to make a routing decision.

- `Queue` — array representing the queue of generated packets that await transmission.

At the beginning of a simulation, the parameters of the simulator are read from the input file. These parameters include the network size N, seeds of random numbers generator, the length of the links, and other more specific parameters that depend on the simulated protocol (like the Hurst parameter of a fractal traffic generator used in a simulation of a videophone). Then, N objects representing switches are created. Depending on the topology and connectivity of the network we want to simulate, the variables described above are initiated. Note that the contents of `Neighbour` define the topology of the network, and its size determines the connectivity.

Then, the shortest path algorithm computes the shortest distances between each pair of switches in the network. A global, NxN array storing these values is used by a routing algorithm executed on the incoming packets at every switch.

The simulator described here is slotted, i.e., simulation time is divided into slots of the same length. During one slot unit of time, the simulator performs four main operations that constitute a single *simulation cycle*:

1. New packets are generated and inserted to the queues at the switches. The number and distribution of new packets depends on the traffic generator used in a simulator.

2. All slots in the links are shifted by one.

3. Every switch:

    - receives the packets that were addressed to it;
    - inserts new packets if there are any in the queue and if some slots are free at the switch;

- routes the packets to outgoing ports according to the used routing algorithm (locally optimal, age, etc.).

4. Performance measures (e.g., throughput, jitter) are updated.

These steps are repeated until certain conditions are met, e.g., the overall throughput observed in the network becomes stable. At the end, the results measured throughout the simulation are written to the output file.

Another important class used in the simulator is `Slot` with the following fields:

- `Seq` — sequential number of the packet. It is important in jitter and throughput measurement.
- `Source` — Id of the switch that sent the given packet.
- `Destination` — Id of the switch to which the packet is sent.
- `HopCount` — Counter of the number of hops which is incremented on every hop. It is used in computing end-to-end delays.

In an empty slot, all these variables have null values. When a packet is generated, they are set to appropriate values. They again are cleared by the receiver of the packet, i.e., by the switch whose `Id` is the same as the slot's `Destination`.

Depending on the considered protocol, we were modifying slightly the simulator described above. For example, when we investigated real-time traffic in deflection networks (Chapter 2), we selected two most distant switches in the network: the source and the receiver. Then, the source was generating packets at a constant rate whose value was one of the simulator parameters. Traffic generators in all other switches except the two selected ones were generating packets according to some distribution (e.g., Poisson). During the simulation, throughput and interarrival delays were computed at the selected receiver. Throughput was computed using the number of received packets and the number of elapsed simulation cycles. Jitter was computed on the basis of delays between arrivals of consecutive packets.

B.2 Asynchronous Simulator

In this section, we describe the asynchronous simulator used in Chapter 7 in which we investigated the effect of transient buffers on the performance of asynchronous deflection networks. The operation of this simulator is based on the notion of discrete events (e.g., arrival or transmission of a packet, etc.) that are scheduled and executed at various times.

Let us start our description from "event" which is represented by class `Event` with the following fields common for all events:

- `Wake` — time at which the event will be executed.
- `SwitchId` — Id of the switch with which the event is associated. For example, the event responsible for inserting a packet to the network must be associated with some switch.

`Event` also contains a virtual function `Process()`. By overloading this function in descendant classes (i.e., particular events) we are able to arbitrarily change the behaviour of the event.

Almost every event completes the execution of function `Process()` by scheduling another event. For example, event `INSERT` whose purpose is to insert a new packet to the input buffer of a switch sets event `SEND` scheduled to be executed `bufferLength` bits (i.e., units of time) from the current moment. After that (simulated) time, the packet reaches the end of the buffer and may take part in a routing decision, i.e., event `SEND` is triggered.

Obviously, the events scheduled for execution must be stored and handled somewhere. For this purpose, we use class `Kernel`. Its most important function is `Root()` that performs essentially three operations:

1. It finds the earliest event on the list of events, i.e., the event whose `Wake` value is the smallest.
2. It advances the simulation time to that moment.
3. It removes the found event from the list and calls the `Process()` function of that event, i.e., the event is executed.

As we have mentioned, an event often creates a new event and schedules its execution, after which the event is inserted to the list. These three steps are executed in a loop until the special event `END` is encountered. `END` is scheduled when the prescribed simulation time has elapsed. This event completes the simulation.

Other features of the simulator are similar to what we have already described in the previous section. That is, performance measures are computed and written to the files, packets are generated according to the prescribed traffic scenarios, etc. Unlike the slotted simulator, various actions of the network are represented by events whose execution order is asynchronous. For example, in the slotted network, all switches in the network route the packets to outgoing links or receive packets addressed to them *at the same time*. In the asynchronous network, whose simulator was described in this section, these two events may occur at different switches at completely different moments.